METODE *LOAD BALANCING HAPROXY* PADA OPENNEBULA

SKRIPSI

Diajukan sebagai salah satu syarat untuk menyelesaikan
pendidikan diploma empat (D-4) Program Studi Teknik Komputer dan Jaringan
Jurusan Teknik Elektro
Politeknik Negeri Ujung Pandang

MUH. SYAHRIR
425 19 040

PROGRAM STUDI D-4 TEKNIK KOMPUTER DAN JARINGAN
JURUSAN TEKNIK ELEKTRO
POLITEKNIK NEGERI UJUNG PANDANG
MAKASSAR
2023

HALAMAN PENGESAHAN

Skripsi dengan judul **"Metode *Load Balancing Haproxy* pada OpenNebula"**, oleh **Muh. Syahrir** NIM 425 19 040 telah diterima dan disahkan sebagai salah satu syarat untuk memperoleh gelar Diploma IV (D-4/S1 Terapan) pada Program Studi Teknik Komputer dan Jaringan Jurusan Teknik Elektro Politeknik Negeri Ujung Pandang.

Makassar, 08 Agustus 2023

Mengesahkan,

Pembimbing I	Pembimbing II,
Ir. Dahlia Nur, M.T.	Drs. Kasim, M.T.
NIP. 19641231 199103 2 003	NIP. 19630620 199103 1 002

Mengetahui,

Koordinator Program Studi
Teknik Komputer dan Jaringan

Eddy Tungadi, S.T.,M.T.
NIP.197908232010121001

HALAMAN PENERIMAAN

Pada hari ini, hari Selasa tanggal 08 Agustus 2023, Tim Penguji Ujian Sidang Skripsi telah menerima dengan baik skripsi oleh mahasiswa : **Muh. Syahrir** NIM **425 19 040** dengan judul **Metode *Load Balancing Haproxy* pada OpenNebula.**

Makassar, 08 Agustus 2023

Tim Penguji Ujian Sidang Skripsi:

1 Irmawati, S.T.,M.T.	Ketua	(.....)
2 Eddy Tungadi,S.T.,M.T.	Sekretaris	(.....)
3 Meylanie Olivya,S.T,.M.T.	Anggota I	(.....)
4 Zawiyah Saharuna,S.T.,M.Eng.	Anggota II	(.....)
5 Ir. Dahlia Nur,M.T.	Pembimbing I	(.....)
6 Drs. Kasim,M.T.	Pembimbing II	(.....)

KATA PENGANTAR

Alhamdulillah wa syukurillah, puji syukur penulis panjatkan kehadirat Allah SWT karena berkat rahmat dan karunia-Nya, penulisan skripsi ini yang berjudul **"Metode *Load Balancing Haproxy* pada OpenNebula.".**

Penulis menyampaikan terima kasih kepada seluruh pihak yang telah memberikan bantuan, dukungan dan motivasi selama studi hingga terselesaikannya skripsi ini, terutama kepada:

1. Kedua orangtua tercinta dan keluarga yang selalu memberikan doa, semangat dan dorongan baik secara moril maupun materil.
2. Ir. Ilyas Mansur, M.T., selaku Direktur Politeknik Negeri Ujung Pandang;
3. Ahmad Rizal Sultan, S.T., M.T., Ph.D., sebagai Ketua Jurusan Teknik Elektro Politeknik Negeri Ujung Pandang;
4. Eddy Tungadi, S.T., M.T. selaku Koordinator Program Studi D 4 Teknik Komputer dan Jaringan Politeknik Negeri Ujung Pandang;
5. Ibu Ir. Dahlia Nur, M.T. sebagai Pembimbing I dan Bapak Drs. Kasim, M.T. sebagai Pembimbing II yang mana keduanya telah memberikan bimbingan, arahan, bantuan serta dorongan kepada kami dengan penuh kesabaran.
6. Seluruh Dosen Jurusan Teknik Elektro yang selama ini dengan ikhlas telah mendidik dan mengajar kami.
7. Saudara-saudari 4B D4 Teknik Komputer dan Jaringan angkatan 2019 yang bersama-sama telah melalui ini dengan suka-duka yang ada dan selalu memberikan bantuan, kerjasama, motivasi dan semangat.

8. Seluruh rekan-rekan mahasiswa Politeknik Negeri Ujung Pandang yang telah meluangkan waktu untuk memberikan bantuan dan motivasi.

Semoga seluruh kebaikan yang diberikan mendapatkan balasan dari Tuhan Yang Maha Esa. Atas segala kekurangan dan kesalahan yang terdapat dalam laporan ini, kami menyampaikan permohonan maaf, serta mengharapkan kritik dan saran untuk perbaikan ke depannya. Semoga laporan ini dapat memberikan ilmu dan manfaat bagi para pembaca.

Makassar, 08 Agustus 2023

Penulis

DAFTAR ISI

DAFTAR GAMBAR

hal.

DAFTAR TABEL

hal.

DAFTAR LAMPIRAN

hal.

SURAT PERNYATAAN

Saya yang bertanda tangan di bawah ini:

Nama : Muh Syahrir

NIM : 42519040

Menyatakan dengan sebenar-benarnya bahwa segala pernyataan dalam skripsi ini yang berjudul "**Metode *Load Balancing Haproxy* pada OpenNebula**" merupakan gagasan dan hasil karya saya sendiri dengan arahan komisi pembimbing, dan belum pernah diajukan dalam bentuk apapun pada perguruan tinggi dan instansi manapun.

Semua data dan informasi yang digunakan telah dinyatakan secara jelas dan dapat diperiksa kebenarannya. Sumber informasi yang berasal atau dikutip dari karya yang diterbitkan dari penulis lain telah disebutkan dalam naskah dan dicantumkan dalam skripsi ini.

Jika pernyataan saya tersebut diatas tidak benar, saya siap menanggung resiko yang ditetapkan oleh Politeknik Negeri Ujung Pandang.

Makassar, 08 Agustus 2023

Muh Syahrir

42519040

METODE *LOAD BALANCING HAPROXY* PADA OPENNEBULA

RINGKASAN

Server tunggal opennebula dapat *down* atau *overload* jika user yang akses secara bersamaan, banyaknya data yang dibuat, banyaknya konsumsi daya, processor kurang dan bahkan *storage* pada server kurang memadai sehingga beban kerja pada server tidak dibagi secara merata.

Tujuan dari penelitian ini agar dapat meminimalisir server *down* akibat peningkatan beban kerja dan meningkatkan kinerja server yang lebih cepat dengan menggunakan *load balancing haproxy* pada server untuk membagi permintaan layanan yang meminta *request* pada server. Algoritma yang digunakan yaitu *least bandwith* adalah algoritma dikonfigurasi menggunakan metode *bandwith* paling sedikit kemudian diteruskan ke server dalam satuan waktu sedangkan *least response time* adalah algoritma yang mengarahkan user ke server dengan *response time* yang terendah, dengan pengujian menggunakan metode *failover* dan parameter pengukuran berdasarkan standar TIPHON yaitu *response time (ms), packet loss/error rate (%)* dan *throughput (Kbps)*.

Berdasarkan Hasil pengukuran *load balancing* dengan *request* user 11100 dengan standar berdasarkan *Telecommunications and Internet Protocol Harmonization Over Networks (TIPHON)* dimana *response time* untuk algoritma *round robin* sebesar 446,89 ms masuk kategori sedang, algoritma *least bandwith* sebesar 402,45 ms kategori sedang dan *least response time* sebesar 393,67 ms masuk dalam kategori baik. Kemudian *packet loss/Error rate* (%) untuk algoritma *round robin* 20,37 %, *least bandwith* 12,65 % dan *least response time* 7,12 % masuk kategori baik berdasarkan standar. Sedangkan *throughput round robin* 2894,1 Kbps, *least bandwith* 3618 Kbps dan *least response time* 3589 Kbps masuk dalam indeks 4 kategori sangat baik

Kata Kunci: *Cloud Computing Opennebula,* Server*, load balancing haproxy, round robin, least bandwith, least response time.*

BAB I PENDAHULUAN

1.1 Latar Belakang

Kemajuan ilmu pengetahuan dan teknologi di bidang penyedia informasi semakin meningkat terutama kebutuhan akan akses internet. Banyak di jumpai penggunaan Internet terutama di Warnet, Kantor-kantor, Sekolah maupun Kampus menggunakan lebih dari satu koneksi dalam berlangganan Internet (Leman, 2019).

Terutama *Cloud Computing* mengalami peningkatan popularitas yang mengesankan baik industri perangkat lunak maupun dunia penelitian. Fitur paling menarik yang dihadirkan *Cloud Computing* dari sudut pandang klien *Cloud* seperti memudahkan dalam manajemen bisnis, biaya cenderung lebih hemat, mempunyai fitur *back-up* dan pemulihan data dan memiliki kapasistas penyimpanan tanpa batas contohnya yang populer adalah *Cloud Computing* OpenNebula (Khair et al., 2021).

OpenNebula adalah sebuah layanan *Cloud Computing* yang menjadi mesin virtual dengan menyediakan grup yang efisien, dinamis, terukur yang berinteraksi dengan server virtual dan fisik. Server tunggal pada opennebula mengalami *down* jika banyak *user* yang akses secara bersamaan, banyaknya data yang dibuat, banyaknya konsumsi daya, processor kurang dan bahkan *storage* pada server kurang memadai. oleh karena itu menawarkan kesederhanaan dalam memberikan solusi yang lengkap untuk membuat dan melakukan manajemen virtualisasi data center dalam tingkat *enterprise* dengan proses *cluster* server *load balancer* (Sudirman, 2020).

Load balancing pada server merupakan salah satu cara yang dapat digunakan untuk meningkatkan kinerja dan ketersediaan server, yaitu dengan membagi

permintaan layanan yang datang ke beberapa server sekaligus, sehingga beban yang ditanggung oleh masing-masing server lebih sedikit (Riskiono & Pasha, 2020). Dalam menerapkan *load balancing haproxy* pada Opennebula permintaan paket dapat diteruskan dari pengguna ke server. Adapun algoritma yang digunakan yaitu *least response time* dan *least bandwith*. Algoritma *least response time* merupakan algoritma yang mengarahkan user ke server dengan *least response time* dan waktu respon terendah. *Least response time* digunakan dimana dua parameter (koneksi paling tidak aktif dan waktu respon terendah) sedangkan algoritma *least bandwith* merupakan algoritma yang dikonfigurasi menggunakan metode *bandwith* paling sedikit kemudian diteruskan ke server dalam megabit per detik (Mbps) (Alankar et al., 2020).

Penelitian sebelumnya yang dilakukan oleh (Thapliyal & Dimri, n.d.) dengan berjudul "*Load Balancing in Cloud Computing Based on Honey Bee Foraging Behavior and Load Balance Min-Min Scheduling Algorithm*". Algoritma yang digunakan adalah min-min *load balancing* untuk menyeimbangkan beban di jaringan komputasi awan dimana menghasilkan hasil komparatif dengan memperhatikan parameter seperti konsumsi daya, waktu respon, skalabilitas dan daya tahan dalam perencanaan sumber daya terintegrasi. Penelitian yang dilakukan oleh (Madakatte & Nagesh, 2021) dengan judul "*Arr And Fusion Based Virtual Machine Scheduling Techniques For Eucalyptus Cloud*" yaitu menggunakan teknik penjadwalan mesin *Eucalyptus* yaitu *Advanced Round-Robin* (*ARR*) dan metodologi Fusion.

Dengan demikian penelitian ini dirancang dengan metode *load balancing haproxy* pada Opennebula menggunakan *least response time* dan *least bandwith* agar supaya web server dapat meminimalisir terjadinya server *down* akibat peningkatan jumlah beban kerja pada server dan meningkatkan kinerja server yang lebih cepat.

1.2 Rumusan Masalah

Berdasarkan latar belakang masalah yang telah dikemukakan, maka rumusan masalahnya adalah sebagai berikut:

1) Bagaimana menerapkan metode *load balancing haproxy* pada OpenNebula?
2) Bagaimana kinerja server pada OpenNebula menggunakan metode *load balancing haproxy*?

1.3 Ruang Lingkup Penelitian

Ruang lingkup penelitian tugas akhir ini adalah:

1) Tugas akhir ini dirancang penerapan dalam bentuk virtualisasi.
2) Menggunakan laptop minimal *Processor* Intel *Core i*5 2.7 GHz, memori 8GB atau Komputer *Processor* Intel *Core i*5 *Harddisk* minimal 500GB dan *RAM* 8GB.
3) Menggunakan Cloud Computing OpenNebula.

1.4 Tujuan Penelitian

Tujuan dari penelitian tugas akhir ini adalah sebagai berikut:

1) Menerapkan metode *load balancing haproxy* pada Opennebula
2) Melakukan pengukuran kinerja server pada OpenNebula menggunakan metode *load balancing haproxy.*

1.5 Manfaat Penelitian

Manfaat penelitian tugas akhir ini adalah:

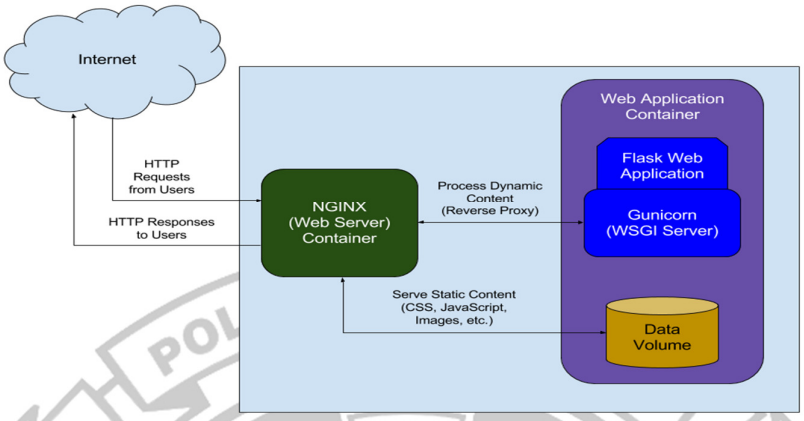
1) Minimalisir terjadinya server *down* akibat peningkatan jumlah beban kerja.

2) Meningkatkan kinerja server yang lebih cepat.

BAB II TINJAUAN PUSTAKA

2.1 Web Server

Web server adalah *machine* yang dapat merespon permintaan user. tanggapannya adalah tentang situs web. Ketika perangkat lunak diinstal dan terhubung ke internet, komputer dapat menjadi server web. Server web bertanggung jawab untuk menerima dan melayani permintaan HTTP dari user web, biasanya di halaman web yang berisi konten statis (teks, gambar, dll.) dan konten dinamis (*script*). Oleh karena itu, peran utama web server adalah untuk menyimpan dan memproses file dan menyediakan user dengan permintaan halaman web melalui protokol komunikasi antara user dan server yang menggunakan *Hypertext Transfer Protocol (HTTP)*. Halaman yang dikirimkan adalah dokumen *Hypertext Markup Language (HTML)*. Banyak server web bergantung pada server web Apache, yang merupakan server web yang paling umum digunakan, salah satu yang paling populer karena Apache dapat digunakan. Ini adalah server web HTTP dan memiliki kontrol akses, PHP, dan dukungan SSL. Server web yang paling populer adalah Apache Web Server, Nginx, dan NodeJS (Ibrahim et al., 2021). Adapun contoh web server seperti Gambar 2.1



Gambar 2.1 Web Server

2.2 Teknologi *Load Balancing*

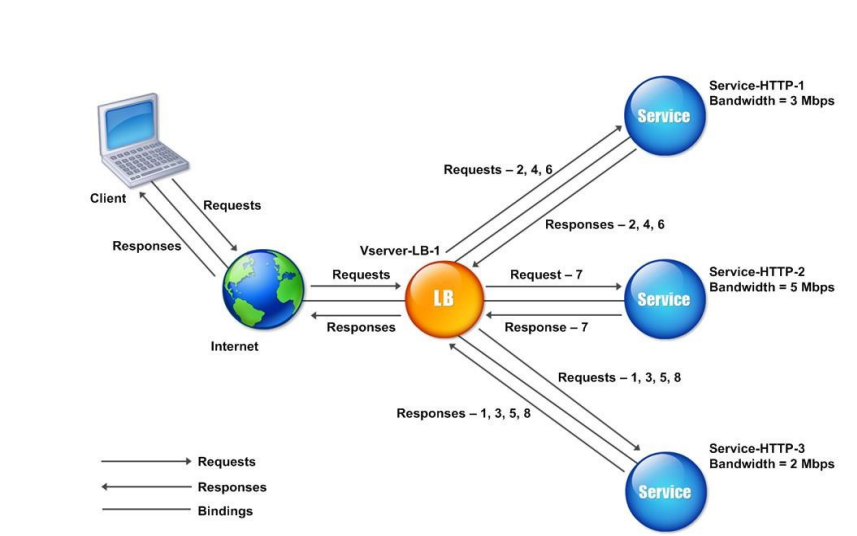
Load balancing bekerja dengan cara mendistribusikan beban trafik yang datang secara merata ke beberapa server – server yang terhubung pada mesin *load balancer*, sehingga beban trafik yang diterima akan lebih ringan untuk dilayani. dengan menggunakan teknik *load balancing* dapat mempersingkat waktu akses terhadap web server dan memiliki ketersediaan layanan yang tinggi. *Load balancing* menghindari hambatan di depan menggunakan berbagai aplikasi, server web *cluster* yang *homogen* dan *heterogen*. Selain itu, *load balancing* server web merupakan kebutuhan penting dalam teknologi web untuk menjamin kualitas layanan yang optimal dengan mengevaluasi metrik kinerja yang dikeluarkan dari status server dan menyeimbangkan beban berdasarkan hasil pengukuran kinerja (Jader et al., 2019).

Adapun algoritma yang digunakan adalah *least response time* dan *least bandwith.*

2.2.1 *Least Bandwith*

Load balancing server yang dikonfigurasi untuk menggunakan metode *bandwith* paling sedikit memilih layanan yang saat ini melayani jumlah lalu lintas paling sedikit, diukur dalam *megabit per detik* (*Mbps*). Contoh berikut menunjukkan bagaimana server virtual memilih layanan untuk penyeimbangan beban dengan menggunakan metode bandwith paling sedikit. Pertimbangkan tiga layanan, layanan-HTTP-1 memiliki *bandwidth* 3 Mbps, layanan-HTTP-2 memiliki *bandwidth* 5 Mbps dan layanan-HTTP-3 memiliki *bandwidth* 2 Mbps (Majumder, 2021a).

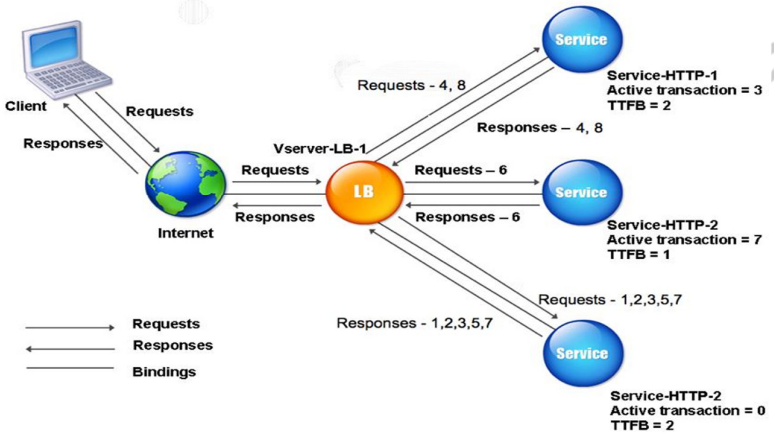
Diagram berikut mengilustrasikan bagaimana server virtual menggunakan metode *bandwidth* paling sedikit untuk meneruskan permintaan ke tiga layanan. Berikut cara kerjanya seperti Gambar 2.2



Gambar 2.2 *Least Bandwidth* (Majumder, 2021a).

2.2.2 *Least Response Time*

Ketika *load balancing* dikonfigurasi untuk menggunakan metode waktu *response* paling sedikit, server ini akan memilih layanan dengan koneksi aktif paling sedikit dan waktu *response* rata-rata terendah. Anda dapat mengkonfigurasi metode ini hanya untuk server virtual penyeimbang beban HTTP dan *Secure Sockets Layer (SSL)*. Waktu respons (juga disebut *Time to First Byte*, atau TTFB) adalah interval waktu antara mengirim paket permintaan ke layanan dan menerima paket respons pertama dari layanan. Contoh berikut menunjukkan bagaimana server virtual memilih layanan untuk penyeimbangan beban dengan menggunakan metode waktu respons paling sedikit. Pertimbangkan tiga layanan berikut: Layanan-HTTP-1 menangani tiga transaksi aktif dan TTFB adalah dua detik, Layanan-HTTP-2 menangani tujuh transaksi aktif dan TTFB adalah satu detik dan Layanan-HTTP-3 tidak menangani transaksi aktif dan TTFB adalah dua detik (Majumder, 2021b). Diagram berikut mengilustrasikan bagaimana *load balancing* menggunakan metode waktu respons paling sedikit untuk meneruskan koneksi seperti Gambar 2.3

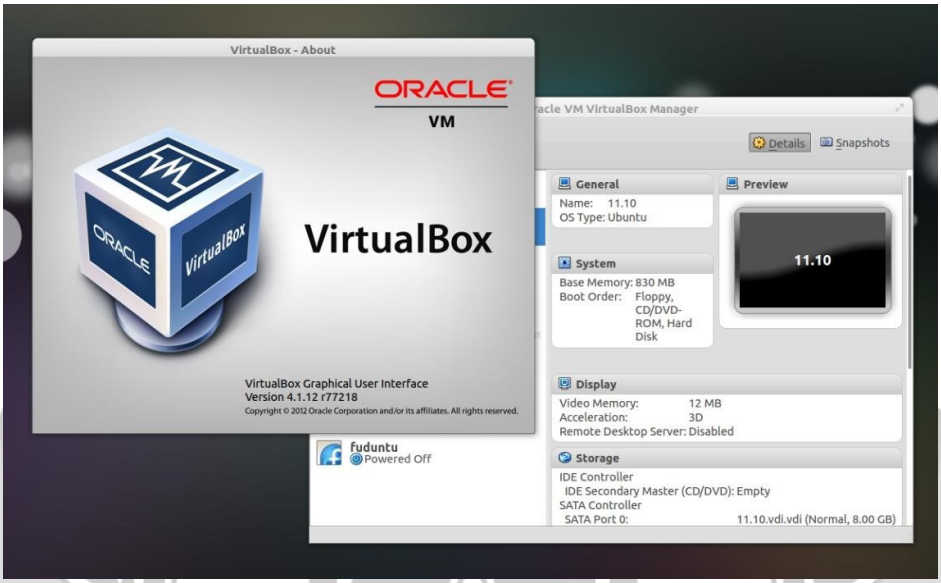


Gambar 2.3 *Least Response Time* (Majumder, 2021b).

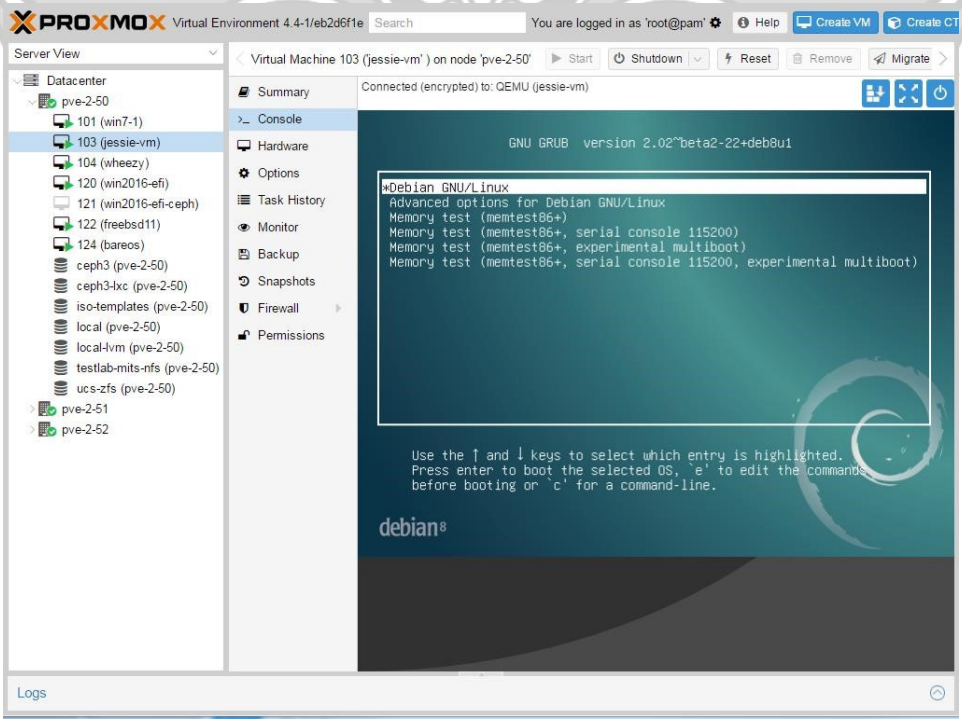
2.3 Teknologi Virtualisasi

Virtualisasi /*Virtualization* adalah sebuah teknik atau cara untuk membuat sesuatu dalam bentuk virtualisasi, tidak seperti kenyataan yang ada (Widarma & Siregar, 2019). Virtualisasi juga digunakan untuk mengemulasikan perangkat fisik komputer, dengan cara membuatnya seolah-olah perangkat tersebut tidak ada (disembunyikan) atau bahkan menciptakan perangkat yang tidak ada menjadi ada.Teknologi virtualisasi ini merupakan sebuah teknologi yang memungkinkan sebuah mesin yang berbentuk fisik dijadikan sebuah sumber daya bersama yang dapat dibagi dan digunakan oleh beberapa layanan sekaligus. Dengan kata lain, virtualisasi adalan proses mengubah perangkat fisik (*hardware*) menjadi perangkat lunak (*software*). Pada masing-masing perangkat virtual juga dapat dikonfigurasi tanpa mempengaruhi satu sama lain walaupun dalam satu lingkungan virtualisasi. Masing-masing perangkat virtual juga dapat memiliki sistem operasi sendiri-sendiri, sehingga konfigrasi dari masing-masing perangkat pun tidak saling mempengaruhi. Selain itu, masing-masing mesin virtual pada lingkungan virtualisasi dapat dimatikan atau dihidupkan sesuai dengan kebutuhan tanpa harus mengganggu layanan lainnya. Sehingga ketersediaan sebuah layanan dapat lebih terjamin meskipun terdapat beberapa layanan yang sedang mengalami masalah atau perbaikan (Widarma & Siregar, 2019). Contoh *software* virtualisasi adalah virtualbox dan proxmox VE. Proxmox VE (*Virtual Environment*) merupakan distro spesial untuk virtualisasi, dibangun dari basis distro debian minimal dan berjalan dalam modus teks. Meski berbasis teks, proses manajemen proxmox virtual *environment* mudah dilakukan melalui akses web termasuk melakukan instalasi

sistem dengan menggunakan teknologi *VNC* (Imanudin, 2018). Contoh virtualisasi virtualbox seperti Gambar 2.4 dan proxmox VE seperti Gambar 2.5



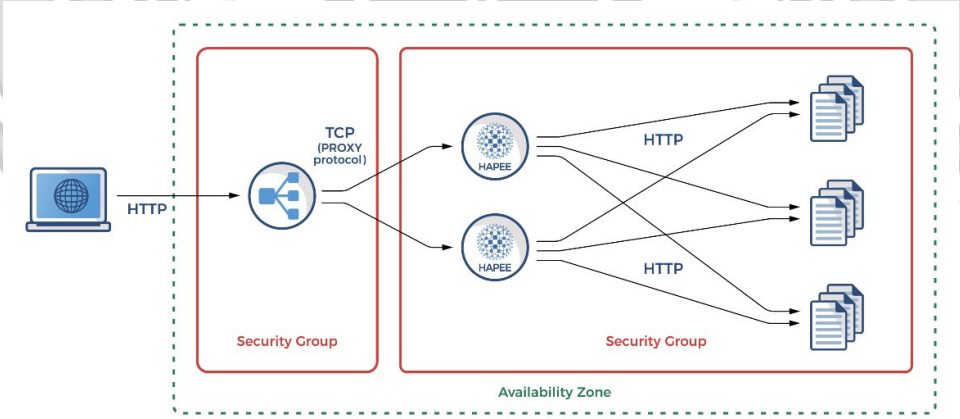
Gambar 2.4 Virtualbox



Gambar 2.5 Proxmox VE

2.4 *Haproxy*

Haproxy menawarkan layanan *load balancing* ke layanan berbasis HTTP dan TCP, seperti layanan yang terhubung ke internet dan aplikasi berbasis web. Bergantung pada algoritma penjadwalan sebagai penyeimbang beban yang dipilih, sehingga *haproxy* dapat melayani ribuan koneksi yang terjadi. Haproxy dipasang pada server *front-end*. *Front-end* server umumnya adalah server yang memiliki IP statis teregistrasi dengan DNS. *Haproxy* digunakan oleh sejumlah situs *high-profile* termasuk StackOverflow, Reddit, Tumblr, dan Twitter dan digunakan dalam produk OpsWorks dari Amazon Web Services (Syamsu, 2018). Untuk lebih jelasnya seperti Gambar 2.6



Gambar 2.6 Haproxy

2.5 OpenNebula

OpenNebula adalah sebuah layanan *cloud* yang dikembangkan untuk kebutuhan virtualisasi pusat data dan menjadi mesin virtual yang menyediakan grup yang efisien, dinamis, terukur yang berinteraksi dengan server virtual dan fisik. (Sudirman, 2020).

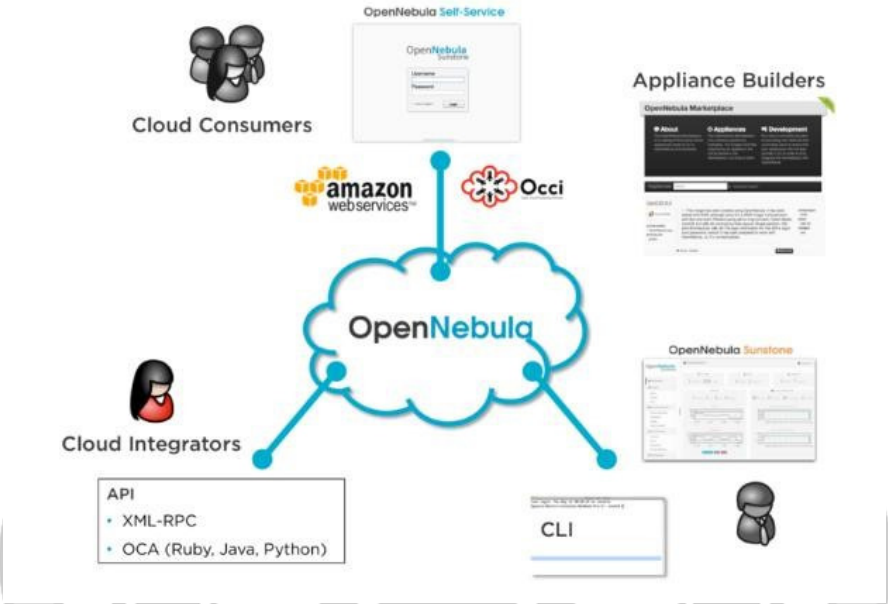
2.5.1 Fitur OpenNebula

OpenNebula memiliki beberapa fitur diantaranya adalah sebagai berikut

a. *Interfaces* yang disediakan oleh OpenNebula

OpenNebula menyediakan banyak *interfaces* berbeda yang dapat digunakan untuk berinteraksi dengan fungsionalitas yang ditawarkan untuk mengelola sumber daya fisik dan virtual. Ada empat perspektif utama yang berbeda untuk berinteraksi dengan OpenNebula (lihat Gambar 2.7).

1. *Cloud Interfces* untuk konsumen *cloud*, seperti antarmuka OCCI dan EC2 Query dan EBS,dan tampilan pengguna *cloud* Sunstone sederhana yang dapat digunakan sebagai portal layanan mandiri.
2. *Administration interfaces* untuk pengguna dan operator tingkat lanjut *cloud*, seperti antarmuka baris perintah mirip Unix dan GUI Sunstone yang kuat.
3. API tingkat rendah yang dapat diperluas untuk *cloud Integrators* di Ruby, JAVA, dan XMLRPC API
4. *Marketplace* untuk pembuat peralatan dengan katalog peralatan virtual yang siap dijalankan di lingkungan OpenNebula.



Gambar 2.7 *Interfaces* OpenNebula

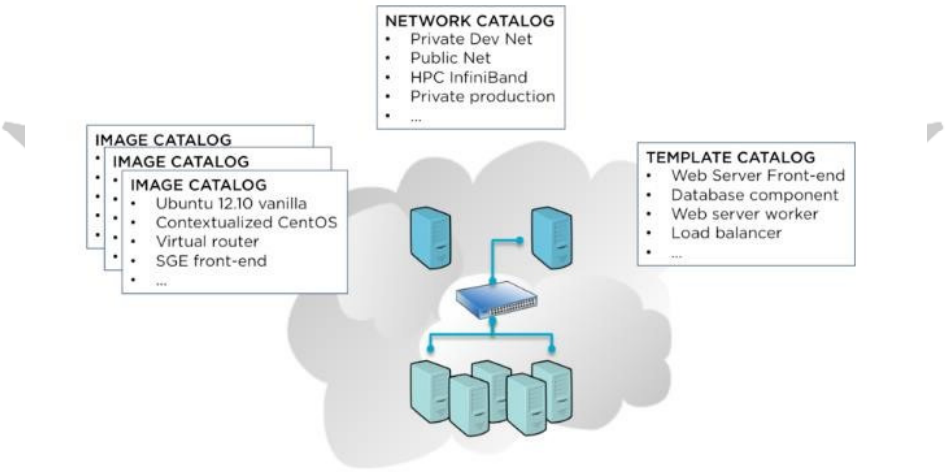
b. OpenNebula kepada Konsumen *Cloud*

OpenNebula menyediakan platform cloud multi-penyewa yang kuat, skalabel dan aman untuk pengiriman cepat dan elastisitas sumber daya virtual. Aplikasi multi-tier dapat digunakan dan digunakan sebagai peralatan virtual pra-konfigurasi dari katalog (lihat Gambar 2.8).

1. *Image Catalogs*: OpenNebula memungkinkan untuk menyimpan gambar disk dalam katalog (disebut penyimpanan data), yang kemudian dapat digunakan untuk mendefinisikan VM atau dibagikan dengan pengguna lain. Gambar dapat berupa instalasi OS, kumpulan data persisten, atau blok data kosong yang dibuat di dalam penyimpanan data.
2. *Network Catalogs*: Jaringan virtual juga dapat diatur dalam katalog jaringan, dan menyediakan sarana untuk menghubungkan mesin virtual. Sumber daya semacam ini dapat didefinisikan sebagai jaringan tetap atau

jarak jauh, dan dapat digunakan untuk mencapai isolasi penuh antara jaringan virtual.

3. *VM Template Catalog*: Sistem katalog template memungkinkan untuk mendaftarkan definisi mesin virtual dalam sistem, untuk digunakan kemudian sebagai instance mesin virtual.
4. *Virtual Resource Control and Monitoring*: Setelah template diinstansiasi ke mesin virtual, ada sejumlah operasi yang dapat dilakukan untuk mengontrol siklus hidup instance mesin virtual, seperti migrasi (langsung dan dingin), *stop*, *resume*, *cancel*, matikan listrik, dll.
5. *Multi-tier Cloud Application Control and Monitoring*: OpenNebula memungkinkan untuk mendefinisikan, menjalankan, dan mengelola aplikasi elastis multi-tier, atau layanan yang terdiri dari mesin virtual yang saling berhubungan dengan ketergantungan penerapan di antara mereka dan aturan penskalaan otomatis .



Gambar 2.8 Konsumen *Cloud* OpenNebula

c. OpenNebula kepada Operator *Cloud*

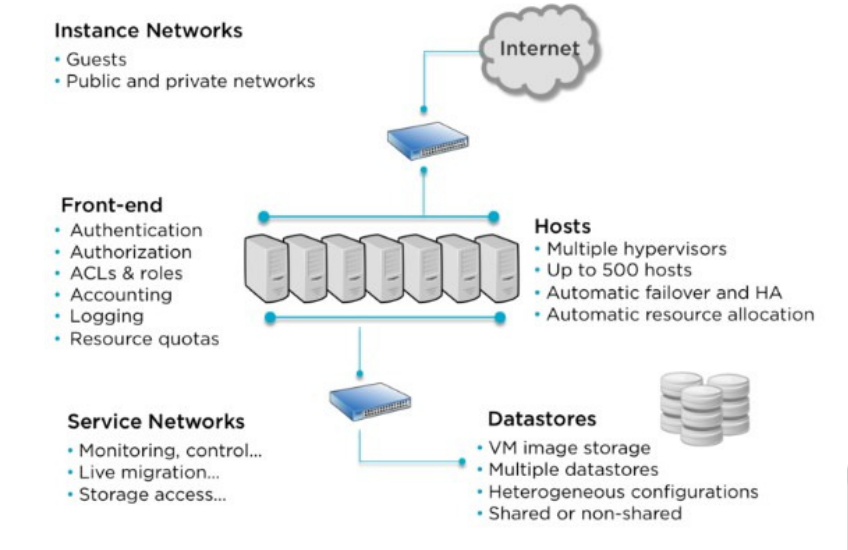
OpenNebula terdiri dari subsistem berikut (lihat Gambar 2.9)

1. *Users and Groups*: OpenNebula menampilkan multi-penyewaan canggih dengan manajemen pengguna dan *grup* yang andal, ACL digunakan alokasi sumber daya, dan manajemen kuota sumber daya untuk melacak dan membatasi penggunaan komputasi, penyimpanan, dan jaringan.
2. *Virtualization*: Berbagai *hypervisor* di dukung di manajer virtualisasi , dengan kemampuan untuk mengontrol siklus hidup lengkap mesin virtual dan beberapa *hypervisor* dalam *infrastruktur cloud* yang sama.
3. *Hosts*: Manajer *host* menyediakan fungsionalitas lengkap untuk pengelolaan *host* fisik di *cloud*.
4. *Monitoring*: Sumber daya virtual serta *host* dipantau secara berkala untuk indikator kinerja utama. Informasi tersebut kemudian dapat digunakan oleh penjadwal yang kuat dan fleksibel untuk definisi beban kerja dan kebijakan alokasi yang sadar sumber daya. Anda juga dapat memperoleh wawasan status dan kinerja aplikasi.
5. *Accounting*: Sistem akuntansi yang dapat dikonfigurasi untuk memvisualisasikan dan melaporkan data penggunaan sumber daya, untuk memungkinkan integrasinya dengan platform tagihan balik dan penagihan, atau untuk menjamin pembagian sumber daya yang adil di antara pengguna.

6. *Networking*: Subsistem jaringan yang mudah beradaptasi dan dapat disesuaikan hadir di OpenNebula untuk lebih berintegrasi dengan kebutuhan jaringan spesifik dari pusat data yang ada dan untuk memungkinkan isolasi penuh antara mesin virtual yang menyusun layanan tervirtualisasi.
7. *Storage*: Dukungan untuk beberapa penyimpanan data dalam subsistem penyimpanan memberikan fleksibilitas ekstrem dalam merencanakan backend penyimpanan dan manfaat kinerja yang penting.
8. *Security*: Fitur ini tersebar di beberapa subsistem: mekanisme otentikasi dan otorisasi yang memungkinkan berbagai mekanisme yang mungkin untuk mengidentifikasi pengguna yang berwenang, mekanisme daftar kontrol akses yang kuat yang memungkinkan manajemen peran yang berbeda dengan pemberian izin butir halus atas sumber daya apa pun yang dikelola oleh OpenNebula, dukungan untuk isolasi pada level yang berbeda.
9. *High Availability*: Dukungan untuk arsitektur HA dan perilaku yang dapat dikonfigurasi jika terjadi kegagalan *host* atau VM untuk menyediakan solusi *failover* yang mudah digunakan dan hemat biaya.
10. *Clusters*: Cluster adalah kumpulan host yang berbagi penyimpanan data dan jaringan virtual. *Cluster* digunakan untuk *load balancing*, ketersediaan tinggi, dan komputasi kinerja tinggi.
11. *Multiple Zones*: Komponen OpenNebula *Zones* (oZones) memungkinkan pengelolaan terpusat dari beberapa instance

OpenNebula, yang disebut *Zones*, untuk skalabilitas, isolasi, dan dukungan multi-situs.

12. *VDCs*: Instance OpenNebula (atau Zone) dapat dikotak-kotakkan lebih lanjut di Pusat Data Virtual (VDC), yang menawarkan lingkungan infrastruktur virtual yang sepenuhnya terisolasi di mana sekelompok pengguna, di bawah kendali *administrator* VDC, dapat membuat dan mengelola komputasi, penyimpanan, dan kapasitas jaringan.
13. *Cloud Bursting*: OpenNebula memberikan dukungan untuk membangun *cloud hybrid*, perpanjangan dari *cloud* pribadi untuk menggabungkan sumber daya lokal dengan sumber daya dari penyedia *cloud* jarak jauh. Seluruh penyedia *cloud* publik dapat dienkapsulasi sebagai sumber daya lokal agar dapat menggunakan kapasitas komputasi ekstra untuk memenuhi permintaan puncak.
14. *App Market*: OpenNebula memungkinkan penyebaran katalog aplikasi *cloud* terpusat pribadi untuk berbagi dan mendistribusikan peralatan virtual di seluruh instance OpenNebula



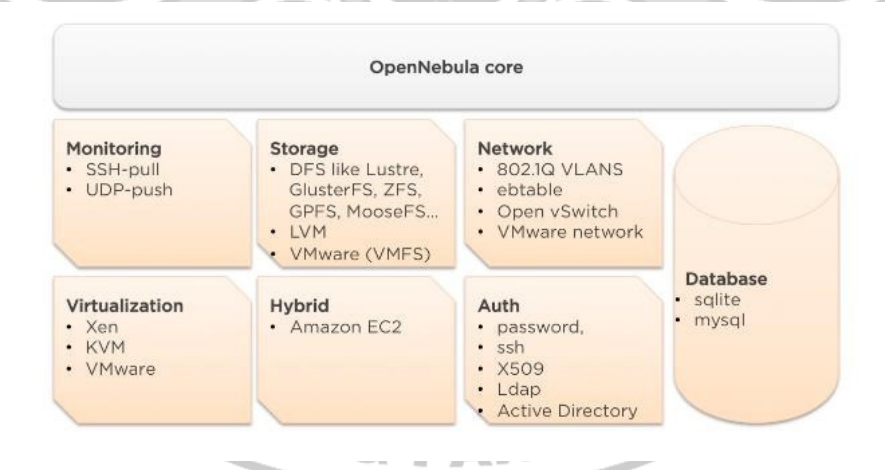
Gambar 2.9 Operator *Cloud* OpenNebula

d. OpenNebula kepada Pembuat *Cloud*

OpenNebula menawarkan dukungan luas untuk komoditas dan *hypervisor* tingkat perusahaan, pemantauan, penyimpanan, jaringan, dan layanan manajemen pengguna (lihat Gambar 2.10)

1. *User Management*: OpenNebula dapat memvalidasi pengguna menggunakan basis data pengguna internalnya sendiri berdasarkan kata sandi, atau mekanisme eksternal, seperti ssh, x509, ldap atau *Active Directory*
2. *Virtualization*: Beberapa teknologi *hypervisor* didukung penuh, seperti Xen , KVM dan VMware .
3. *Monitoring*: OpenNebula menyediakan sistem pemantauannya sendiri yang dapat disesuaikan dan sangat skalabel dan juga dapat diintegrasikan dengan alat pemantauan pusat data *eksternal*.

4. *Networking*: Jaringan virtual dapat didukung oleh 802.1Q VLAN, ebtables, Open vSwitch atau jaringan VMware.
5. *Storage*: Beberapa *back-end* didukung seperti penyimpanan data sistem file biasa (bersama atau tidak) yang mendukung sistem file terdistribusi populer seperti NFS, Lustre, GlusterFS, ZFS, GPFS, MooseFS. VMware datastore (baik sistem file biasa atau berbasis VMFS) khusus untuk *hypervisor* VMware yang menangani format vmdk; penyimpanan data LVM untuk menyimpan gambar disk dalam bentuk perangkat blok dan Ceph untuk perangkat blok terdistribusi.
6. *Databases*: Selain *backend* sqlite asli, mysql juga didukung.
7. *Cloud Bursting*: Konektor bawaan dikirim untuk mendukung cloudbursting Amazon EC2.

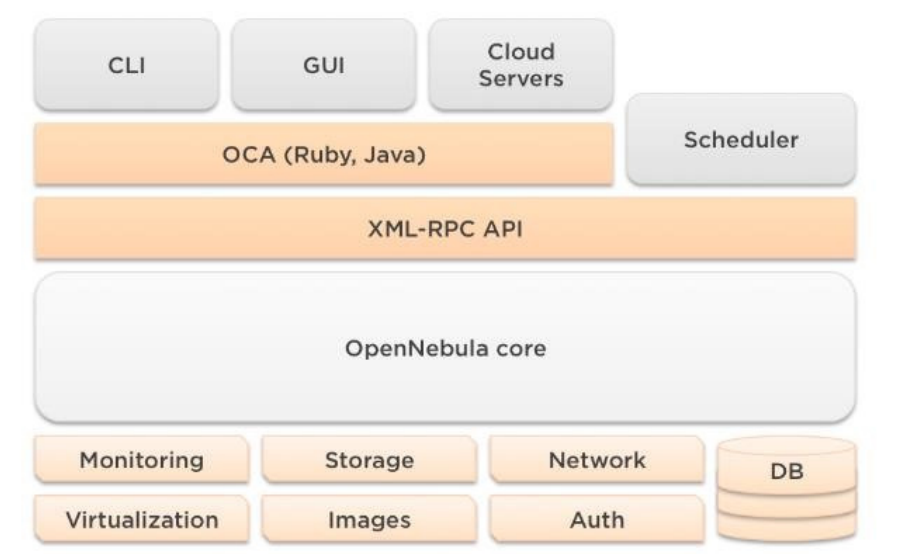


Gambar 2.10 Pembuat *Cloud* OpenNebula

e. OpenNebula kepada *Cloud Integrators*

OpenNebula sepenuhnya independen terhadap platform dan menawarkan banyak alat untuk integrator *cloud* (lihat Gambar 2.11)

1. Arsitektur modular dan dapat diperluas dengan plug-in yang dapat disesuaikan untuk integrasi dengan layanan pusat data pihak ketiga mana pun
2. API untuk integrasi dengan alat tingkat yang lebih tinggi seperti penagihan, portal swalayan. yang menawarkan semua fungsionalitas kaya inti OpenNebula, dengan binding untuk ruby dan java.
3. *oZones API* digunakan untuk mengelola Zona OpenNebula dan Pusat Data Virtual secara terprogram.
4. Rute kustom Sunstone Server untuk memperpanjang server Sunstone.
5. *OneFlow API* untuk membuat, mengontrol, dan memantau aplikasi atau layanan multi-tingkat yang terdiri dari Mesin Virtual yang saling berhubungan.
6. Kaitkan Manajer untuk memicu skrip administrasi saat status VM berubah.



Gambar 2.11 *Cloud Integrators* OpenNebula

Penjelasan fitur OpenNebula diambil dari referensi (OpenNebula, n.d.)

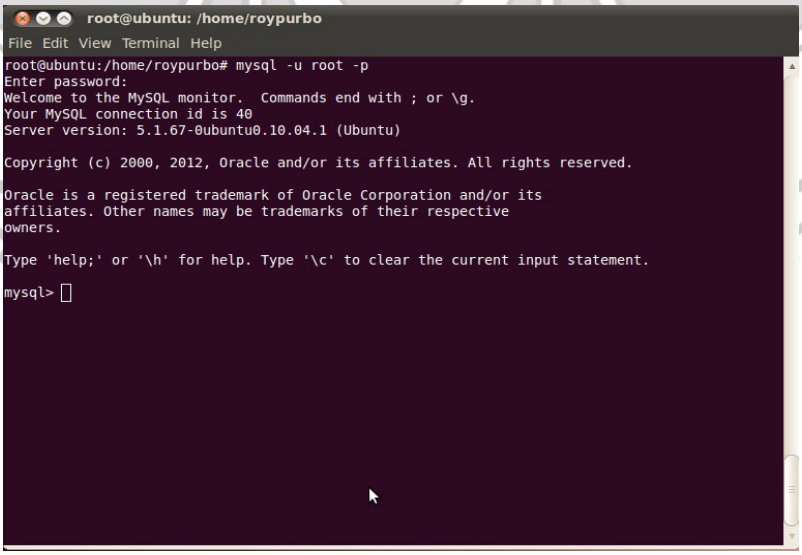
2.6 Database Server

Database server adalah suatu program komputer yang menyediakan layanan pengelolaan dan penyimpanan dengan basis data yang menggunakan model klien server. Dalam database server terdapat sistem yang membantunya bekerja, nama sistem tersebut adalah *Database Management System* (DBMS) (Hartawan, 2021). Sistem ini membantu menyediakan fungsi-fungsi server berbasis data yang berguna untuk menyambungkan antara database dengan pengguna atau user. Dengan begitu database server dapat langsung berinteraksi dalam waktu bersamaan untuk menjalankan tugasnya. Misalnya dalam membuat data, *update*, *ekspor*, query data dan lain sebagainya. Istilah database server sendiri sebenarnya mengacu kepada sebuah komputer yang penggunaannya dikhususkan untuk menjalankan suatu program yang bersangkutan. Adapun macam – macam database server seperti oracle, firebird, Microsoft SQL server 2000, Database desktop paradox, Postgre SQL, Microsoft Access, MySQL dan lainnya.

MySQL adalah sebuah DBMS (*Database Management System*) menggunakan perintah SQL (*Structured Query Language*) yang banyak digunakan saat ini dalam pembuatan aplikasi berbasis website (Adani, 2020). MySQL dibagi menjadi dua lisensi, pertama adalah *Free Software* dimana perangkat lunak dapat diakses oleh siapa saja. Dan kedua adalah Shareware dimana perangkat lunak berpemilik memiliki batasan dalam penggunaannya. MySQL termasuk ke dalam RDBMS (*Relational Database Management System*). Sehingga, menggunakan tabel, kolom, baris, di dalam struktur database -nya. Jadi, dalam proses pengambilan data menggunakan metode relational database. Dan juga menjadi penghubung antara

perangkat lunak dan database server. Kemudian database ini bersifat *Portabilitas*, dimana MySQL bisa berjala stabil di berbagai OS seperti Linux, Amiga, Solaris, Mac Os X Server, FreeBSD, dan lain sebagainya.

Cara kerja pada MySQL yaitu database server menggunakan model klien server, dimana penggunaan model ini mengartikan bahwa sistem membagi prosesnya antara server yang mengolah dan dengan klien yang menjalankan aplikasi. Dengan begitu database server sebenarnya mengurangi beban akses data oleh klien atau pengguna kepada server. Database tersebut dapat diakses oleh beberapa pengguna dalam waktu yang bersamaan dimana data tersebut diakses atau diubah dari satu sumber yang sama yakni pada database server. Server ini dapat diakses baik melalui *front-end* yang sedang berjalan di komputer pengguna yang menampilkan data yang diminta atau melalui *back-end* yang berjalan pada server dan menangani tugas lainnya seperti analisis dan penyimpanan data. Contoh gambar database mySQL di linux seperti Gambar 2.12



Gambar 2.12 MySQL Linux Ubuntu

2.7 *Quality of Service (QoS)*

Kemampuan *QOS* mengacu pada tingkat kecepatan dan kehandalan penyampaian berbagai jenis beban data di dalam sebuah komunikasi (Hikmah et al., 2023). Kualitas jaringan bervariasi berdasarkan beberapa faktor seperti *throughput* dan *response time*. *Quality of Service (QoS)* adalah sebuah metode pengukuran tentang seberapa baik jaringan dan merupakan suatu usaha untuk mendefinisikan karakteristik dan sifat dari satu servis. QoS digunakan untuk mengukur sekumpulan atribut kinerja yang telah dispesifikasikan dan diasosiasikan dengan suatu servis. QoS adalah teknologi jaringan yang memiliki latensi dan kehilangan data yang dapat diprediksi. Lebih khusus lagi, mekanisme apapun yang memungkinkan persyaratan kinerja absolut atau relatif untuk ditentukan untuk aliran lalu lintas yang berbeda yang dibawa melalui jaringan (Dhika dan Syafitri Ayuing Tyas, 2020).

Pada Tabel 2.1 menunjukkan nilai indeks presentase *QoS* dimana setiap indeks presentase menunjukkan level yang berbeda.

Tabel 2.1 Indeks Parameter *QoS*

Kategori	Presentase	Indeks
Buruk	25 – 49,75	1 – 1,99
Sedang	50 – 74,75	2 – 2,99
Baik	75 – 94,75	3 – 3,79
Sangat Baik	95 - 100	3,8 - 4

(Sumber: TIPHON)

Terdapat beberapa parameter sesuai standar *Telecommunications and Internet Protocol Harmonization Over Networks* (TIPHON) untuk mengukur kinerja layanan baik server maupun jaringan internet yaitu:

2.7.1 *Throughput*

Throughput merupakan kecepatan transfer data yang dikirim untuk suatu titik jaringan atau dari suatu titik ke titik jaringan lainnya dalam satuan bps (*bit per second*)

Besarnya *throughput* dapat diperolah dalam persamaan (1) sebagai berikut:

$$Throughput = \frac{jumlah\ data\ yang\ dikirim}{waktu\ pengiriman\ data} \quad (1)$$

Berdasarkan standar TIPHON nilai *throughput* dikategorikan menjadi empat bagian yang dapat dilihat pada Tabel 2.2 berikut ini:

Tabel 2.2 Kategori *Throughput*

Kategori Throughput	Besar Throughput (bps)	indeks
Buruk	<25	1
Sedang	50	2
Baik	75	3
Sangat Baik	100	4

(Sumber: TIPHON)

2.7.2 *Latency/Response Time*

Latency/Response Time merupakan parameter QoS yang menunjukkan waktu yang dibutuhkan paket untuk mendapatkan jarak dari sumber ke tujuan. *Latency* pada suatu jaringan internet dapat dijadikan acuan untuk menilai kualitas dalam proses transmisi data (Maulana dan Pirdania, 2020).

Kemampuan memproses permintaan dan mengirimkan respons atau parameter ini merupakan waktu minimum yang dibutuhkan dari sebuah sistem yang menerapkan algoritma penyeimbangan beban tertentu untuk merespon. Semakin kecil nilai waktu respon maka semakin bagus pula kinerja *system*.

Berdasarkan *Telecommunications and Internet Protocol Harmonization Over Networks (TIPHON)* besarnya *latency* dapat dilihat pada persamaan (2) sebagai berikut:

$$Latency = \frac{Total\ Latency}{Jumlah\ Total\ Packet} \quad (2)$$

Response Time versi TIPHON dapat dikelompokkan menjadi empat kategori seperti yang terlihat pada tabel 2.2 sebagai berikut:

Tabel 2.3 Kategori *Latency*

Kategori Latency	Besar Latency (ms)	Indeks
Buruk	> 450	1
Sedang	> 350 - 450	2
Baik	> 150 - 300	3
Sangat Baik	≤ 150	4

(Sumber: TIPHON)

2.7.3 *Error Rate/Packet Loss*

Packet loss merupakan salah satu parameter dari QoS yang menggambarkan suatu keadaan yang menunjukkan total paket yang gagal atau hilang. *presentase* permintaan yang gagal diterima oleh server atau parameter yang menunjukkan jumlah koneksi yang tidak berhasil dibuat karena gagal dieksekusi atau diproses oleh server *back-end* dan direpresentasikan dalam persen. *Persentase error* dapat dilihat pada kolom *Error* pada tab Aggregate *Report* dari *software* benchmark Apache JMeter.Terdapat beberapa faktor yang mengakibatkan terjadinya kegagalan transmisi saat proses pengiriman paket adalah sebagai berikut:

1. *Overload* trafik yang terjadi pada jaringan
2. Terjadinya *Congestion* di dalam jaringan
3. Terjadi error pada server disebabkan fisik
4. Kegagalan yang terjadi pada sisi penerima yang disebabkan karena *overflow* pada *buffer*

Di dalam implementasi jaringan yang dilakukan, nilai *packet loss* diharapkan mempunyai nilai yang minimum. Adapun persamaan matematis untuk mendapatkan nilai *packet loss* dapat dilihat pada persamaan (3) sebagai berikut:

$$Packet\ Loss = \frac{Paket\ Data\ Dikirim - Paket\ Data\ Diterima}{Paket\ Data\ Dikirim} \quad (3)$$

Berdasarkan standar TIPHON nilai packet loss dibagi menjadi empat indeks yang dapat dilihat pada tabel 2.4 berikut ini:

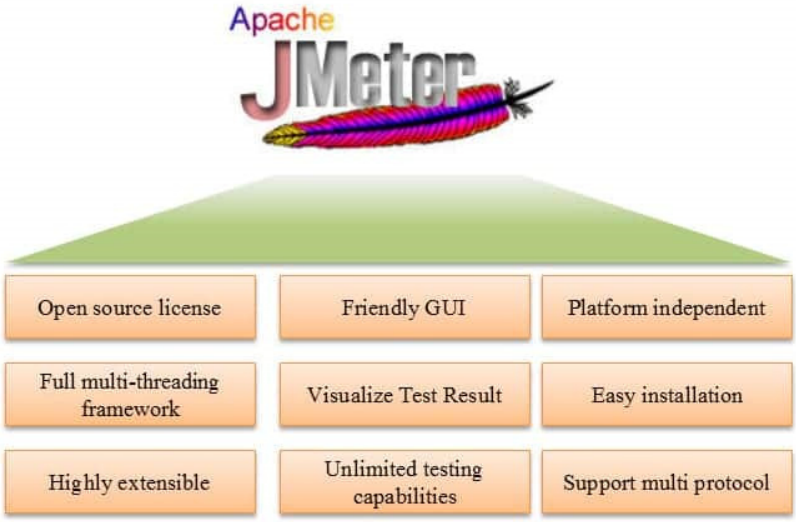
Tabel 2.4 Kategori *Packet Loss*

Kategori *Packet Loss*	Besar *Packet Loss* (%)	Indeks
Buruk	25	1
Sedang	15	2
Baik	3	3
Sangat Baik	0	4

(Sumber: TIPHON)

2.8 Jmeter

Apache Jmeter adalah aplikasi *open source* berbasis java yang menyediakan fitur untuk melakukan *performance test* semua parameter (Prasetia et al., n.d.). Fitur Apache Jmeter seperti pada Gambar 2.13

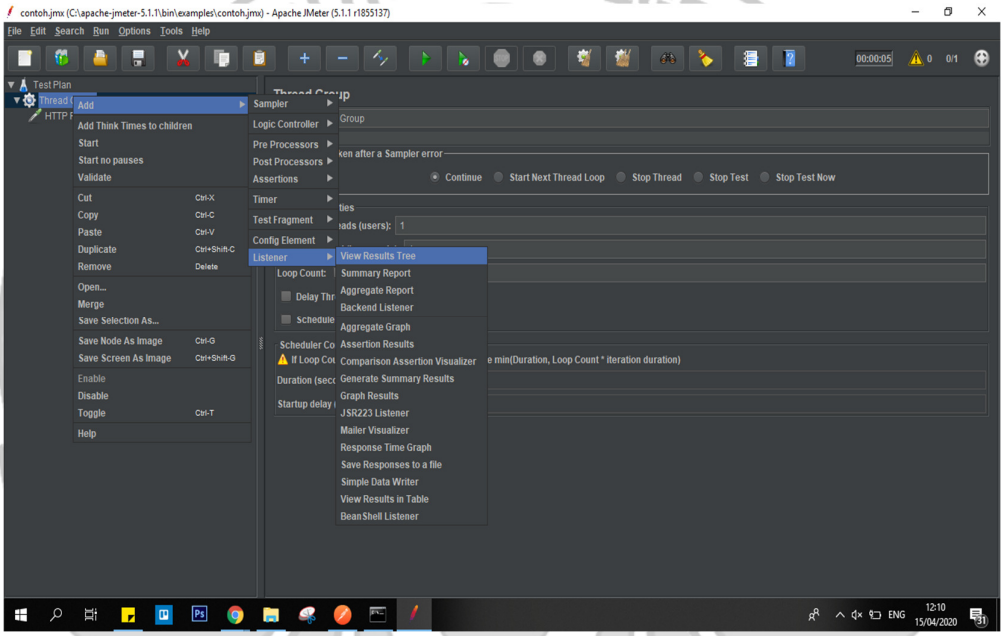


Gambar 2.13 Fitur Apache Jmeter

1. Sebuah GUI yang ramah, Mudah digunakan dan tidak perlu waktu lama untuk membiasakan diri dengan antarmuka program.

2. Platform independen. Programnya adalah Java oleh karena itu, ini dapat berjalan di berbagai platform.
3. *Multi-threading*. JMeter memungkinkan pengambilan sampel simultan dari berbagai fungsi oleh grup utas yang berbeda.
4. Hasil tes dapat dilihat dalam format yang berbeda seperti grafik, tabel, pohon, dan file *log*. Sangat bisa dikembangkan. JMeter juga mendukung plugin tampilan yang memungkinkan kami memperluas pengujian kami.
5. Strategi pengujian ganda. JMeter mendukung banyak strategi pengujian, seperti pengujian beban, pengujian terdistribusi, dan pengujian fungsional.
6. JMeter juga memungkinkan pelaksanaan tes didistribusikan antara komputer yang berbeda, yang akan bertindak sebagai klien.
7. Simulasi. Aplikasi ini dapat mensimulasikan banyak pengguna dengan utas simultan, buat beban berat terhadap aplikasi web yang diuji.
8. Dukungan banyak protokol. Tidak hanya mendukung pengujian aplikasi web, itu juga mengevaluasi kinerja server database. Semua protokol dasar seperti HTTP, JDBC, LDAP, SOAP, JMS, FTP, TCP, dll. Kompatibel dengan JMeter.
9. Rekam dan putar rekam aktivitas pengguna di browser.
10. Tes Skrip. JMeter dapat diintegrasikan dengan Bean Shell dan Selenium untuk pengujian otomatis.
11. Lisensi sumber terbuka. Program ini benar-benar gratis. Jika kita ingin mengetahui kode sumber atau lebih mendalam tentang karakteristik aplikasi ini.

Apache Jmeter bukanlah sebuah browser. *Tools* ini berfungsi pada level protokol. Dari sisi *web-service*, *tools* ini terlihat seperti browser (atau lebih tepatnya beberapa browser) namun sesungguhnya Jmeter tidak melakukan semua tindakan layaknya browser pada umumnya. Secara khusus, Jmeter tidak mengeksekusi javascript yang ditemukan di halaman HTML (Gambar 2.14)



Gambar 2.14 Tampilan Apache Jmeter

BAB III METODE PENELITIAN

3.1 Tempat dan Waktu Penelitian

Penelitian ini dilakukan di Laboratorium Jaringan Komputer Politeknik Negeri Ujung Pandang dengan waktu penelitian dilaksanakan pada bulan Januari 2023 sampai dengan bulan Juni 2023.

3.2 Alat dan Bahan

Beberapa alat dan bahan yang digunakan pada penelitian ini dikelompokkan dalam dua kategori, yaitu perangkat keras (*hardware*) dan perangkat lunak (*software*).

a. Kebutuhan Perangkat Keras (*hardware*)

Perangkat keras yang digunakan dapat dilihat pada Tabel 3.1

Tabel 3.1 Kebutuhan Perangkat Keras (*hardware*)

No.	Perangkat Keras	Keterangan
1.	Laptop: a. Minimal *Processor* Intel Inside 2.7 GHz b. *Memory* 4GB, HDD 500 GB	Digunakan dalam proses konfigurasi sistem
2.	Laptop (Server): a. Minimal *Processor* Intel *Core i5* b. *RAM* minimal 8 GB c. Harddisk minimal 500 GB	Digunakan sebagai tempat konfigurasi server

b. Kebutuhan Perangkat Lunak (*software*)

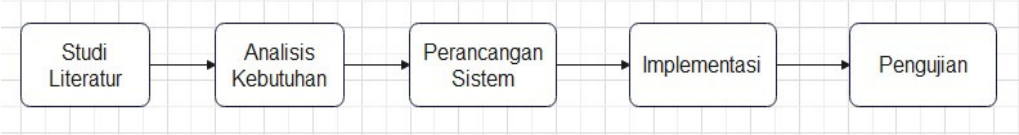
Sedangkan perangkat lunak yang digunakan dapat dilihat pada Tabel 3.2

Tabel 3.2 Kebutuhan Perangkat Lunak (*software*)

No.	Perangkat Lunak	Keterangan
1.	Sistem Operasi *Windows* 10 64 *Bit*	Sistem operasi yang digunakan untuk membangun sistem ini.
2.	Sistem Operasi *Debian 9/* ubuntu 16.04 atau 20.04	Sistem operasi yang digunakan untuk menjalankan *service* yang ada pada sistem dan sebagai server
4.	*Haproxy*	Sebagai perangkat lunak dalam mendistribusikan atau membagikan trafik ke beberapa server
5.	Virtualbox/Proxmox	Sebagai perangkat lunak untuk membuat virtualisasi
6.	OpenNebula	Digunakan untuk membuat cloud computing

3.3 Prosedur Penelitian

Prosedur penelitian diperlukan agar penelitian dapat terstruktur sehingga menghasilkan penelitian yang sesuai dengan tujuan penelitian. Adapun prosedur penelitan yang digunakan adalah metode *waterfall*, model ini berkembang secara sistematis dari satu tahap ke tahap berikutnya. Metode *waterfall* adalah sebuah pendekatan pengembangan *software* yang sistematis dan sekuensial yang dimulai dari tingkat kemajuan sistem analisis, desain, kode program, pengujian dan pemeliharaan. Adapun metode pada penelitian ini dapat dilihat pada Gambar 3.1



Gambar 3.1 Prosedur Penelitian

Berdasarkan tahapan metode penelitian yang akan dilakukan pada Gambar 3.1 maka dapat diuraikan sebagai berikut:

3.3.1 Studi Literatur

Studi literatur yang dilakukan dalam prosedur penelitian ini dengan menggunakan *Library Research* yang merupakan tahapan untuk pengumpulan data dari beberapa buku, jurnal, skripsi, tesis maupun literatur lainnya yang dapat dijadikan acuan pembahasan dalam masalah ini. Penelitian ini memiliki keterkaitan pada sumber-sumber data *online* atau internet ataupun hasil dari penelitian sebelumnya sebagai bahan referensi bagi peneliti selanjutnya.

3.3.2 Analisis Kebutuhan

Kebutuhan sistem yang digunakan untuk penelitian ini dibagi menjadi dua jenis yaitu software dan hardware yang nantinya digunakan dalam persiapan instalasi dan proses konfigurasi seperti Tabel 3.1 dan 3.2

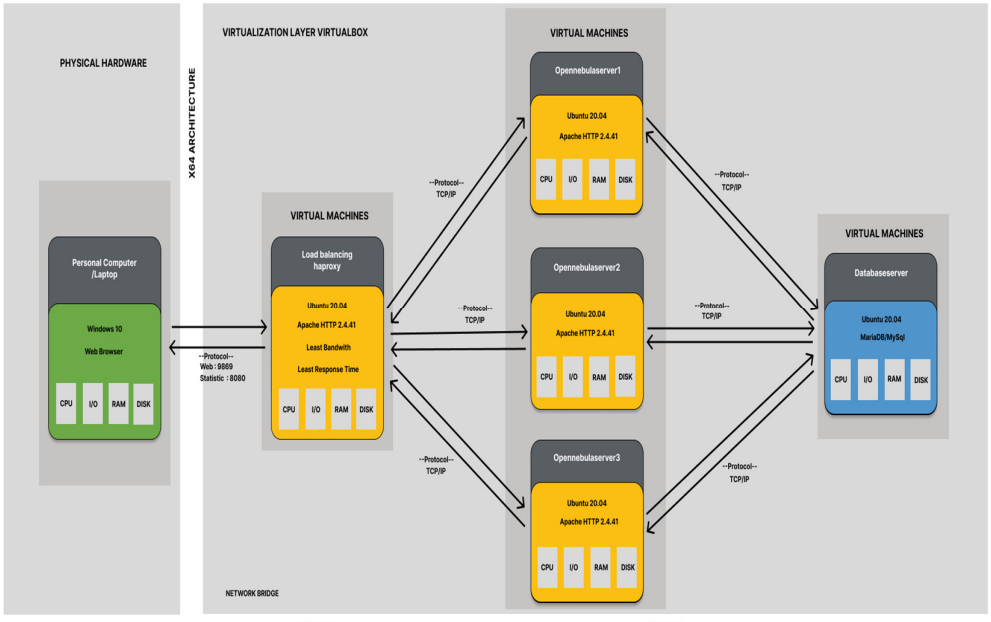
3.3.3 Perancangan Sistem

Tahap perancangan sistem merupakan tahap dalam menggambarkan detail sistem untuk membantu mengidentifikasi informasi yang diperlukan.

a. Desain Sistem Server Opennebula

Proses yang terjadi pada sistem OpenNebula dimana diakses beberapa user atau *client* secara bersamaan dengan jumlah banyak menggunakan akses internet,

untuk menuju ke server opennebula, terlebih dahulu yang harus dilewati adalah server load balancing dengan menggunakan apache HTTP 2.4.41 yang dimana diterapkan 2 algoritma yaitu *least bandwith* dan *least response time* kemudian akan membagi beban atau mendistribusikan trafik ke masing - masing server yang tersedia menggunakan protocol TCP/IP. Server di bagi atas 3 bagian agar supaya tidak memakan banyak beban seperti CPU, RAM dan lain - lain. Kemudian untuk databasenya terpisah agar supaya tidak terbebani di web server, untuk akses opennebula dengan menggunakan protocol HTTP di port 9869 dan *statistic* algoritma yang berjalan pada *haproxy* menggunakan port 8080. Untuk lebih jelasnya dapat dilihat pada Gambar 3.2

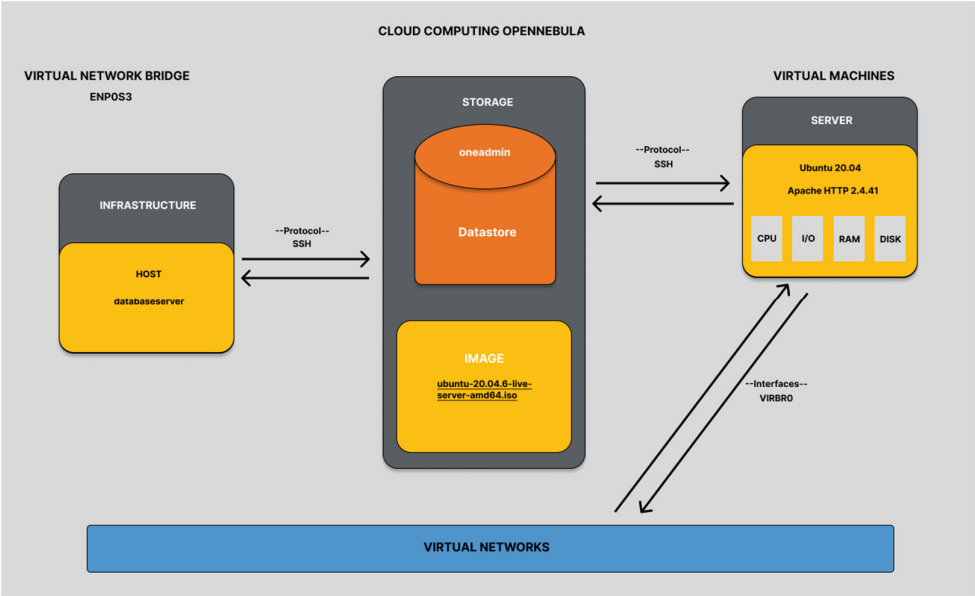


Gambar 3.2 Desain Sistem Opennebula

b. Desain Sistem Server *Cloud Computing*

Pada desain sistem *virtual machines* di *cloud computing* opennebula terdapat virtual *networks* yang terhubung antara *guest* dan *host* dengan

mempunyai 2 *interfaces* dengan ip *address* yang berbeda. Untuk *host* sebagai *infrastructure* yaitu databasenya dengan *storage* sebagai *datastore* penyimpanan *image* dan HDD dan lain sebagainya. Apache yang digunakan yaitu apache HTTP 2.4.41 dengan image ubuntu-20.04.6-live-server-amd64.iso untuk protokolnya menggunakan SSH. Untuk lebih jelasnya dapat dilihat pada Gambar 3.3

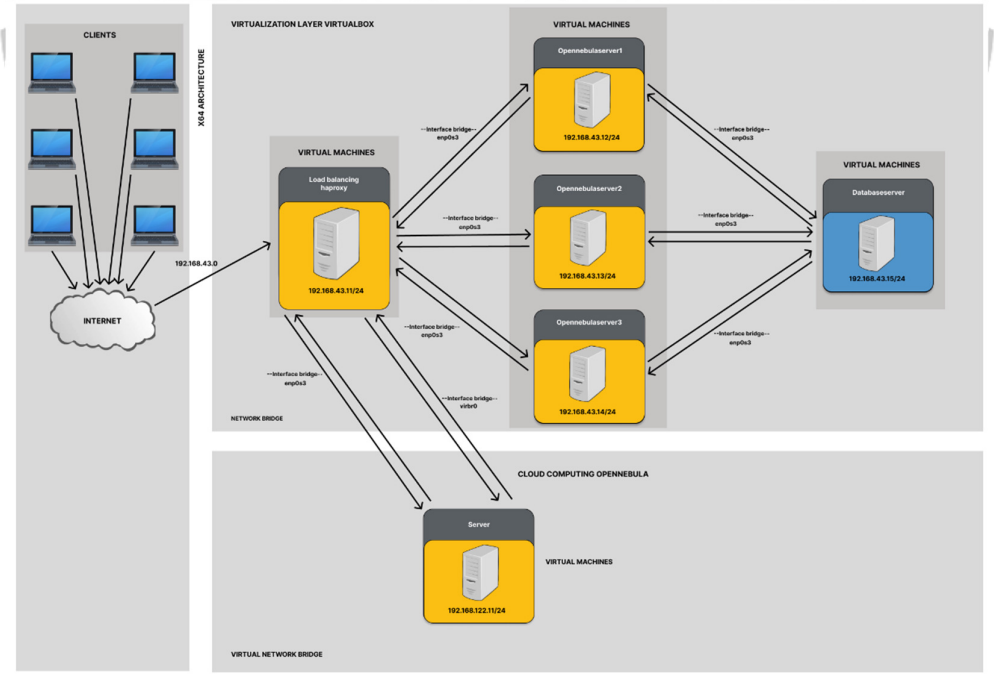


Gambar 3.3 Desain Sistem Server *Cloud*

c. Skema Jaringan

Skema Jaringan menjelaskan bagaimana cara kerja yang dilakukan oleh *load balancing haproxy* pada opennebula berdasarkan denga ip address yang digunakan. Pertama - tama banyak user yang mengakses internet kemudian setelah itu *download* dan install virtualbox sebagai virtualisasi dengan adanya koneksi internet yang memadai. Selanjutnya di dalam virtualbox buat 4 server dan di dalam *cloud computing* opennebula buat 1 server sebagai skenario. Adapun

server yang dibuat yaitu *load balancing haproxy*, opennebulaserver1, opennebulaserver2, opennebulaserver3 dan databaseserver. Kemudian untuk di *cloud* buat 1 server dengan nama server 1. Di dalam virtualbox menggunakan *bridge adapter*, *Bridged adapter* ini memungkinkan OS *guest* untuk menerima data maupun mengirimkan data ke jaringan fisik. Jadi artinya OS *guest* dan OS *host* adalah dua komputer berbeda yang terhubung ke dalam jaringan yang sama. Bila OS *host* memiliki lebih dari satu *Ethernet* maka harus menyetting ke jaringan *virtual machine*/OS *guest* akan disambungkan dan IP yang diberikan ke *virtual machine* harus dari *subnet* yang sama dengan jaringan yang dipakai oleh OS *host*. *Interfaces* yang digunakan yaitu *enp0s3* Untuk lebih jelasnya dapat dilihat Gambar 3.4



Gambar 3.4 Skema Jaringan

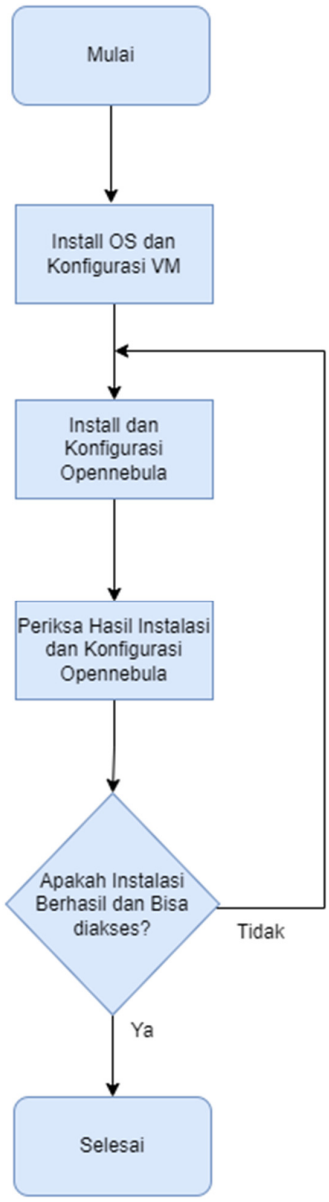
Adapun IP address yang digunakan adalah sebagai berikut

1. Setting IP di VirtualBox
    - Load Balancing Haproxy (192.168.43.11/24)
    - Opennebulaserver1 (192.168.43.12/24)
    - Opennebulaserver2 (192.168.43.13/24)
    - Opennebulaserver3 (192.168.43.14/24)
    - Databaseserver (192.168.43.15/24)
2. Setting IP di *Cloud Computing* Opennebula
    - Server 1 (192.168.122.11/24)

d. *Flowchart* Install dan Konfigurasi Opennebula

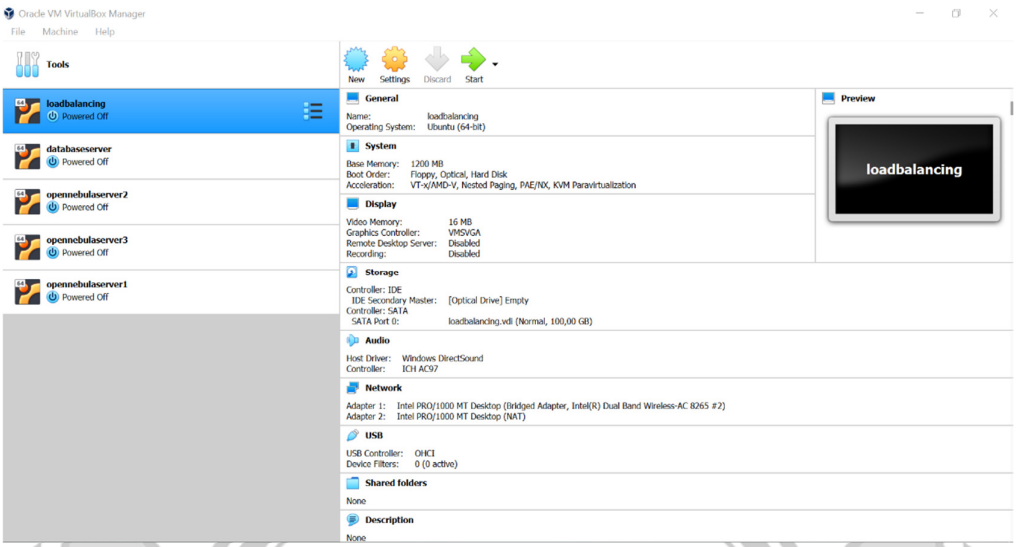
Flowchart ini menjelaskan bagaimana cara install dan konfigurasi yang dilakukan opennebula di virtualbox. untuk lebih jelas memahami cara kerja tersebut maka dibuat dengan menggunakan *flowchart* seperti Gambar 3.5.

Mekanisme install dan konfigurasi Opennebula seperti pada Gambar 3.5 yaitu dengan membuat virtual mesin atau sering disebut sebagai VM di virtualbox. Dalam pembuatan VM Adapun OS yang dipakai adalah Ubuntu 20.01.5-live-server-amd4.iso seperti pada Gambar 3.6. Dimana masing-masing server mempunyai kapasitas yang sama untuk RAM yaitu 2 GB kemudian untuk HDD nya yaitu 100 GB kemudian menggunakan mode *network NAT* dan *Bridge* yang mempunyai satu subnet jaringan yang sama.



Gambar 3.5 Flowchart Install dan Konfigurasi Opennebula

Dalam settingan *system* pada VM di bagian *acceleration* pada *paravirtualization interface* di setting dengan KVM dan *hardware virtualization* di aktifkan agar supaya ketika melakukan *create* VM pada *cloud computing* opennebula bisa berjalan dengan lancar Ketika VNC server *build*.

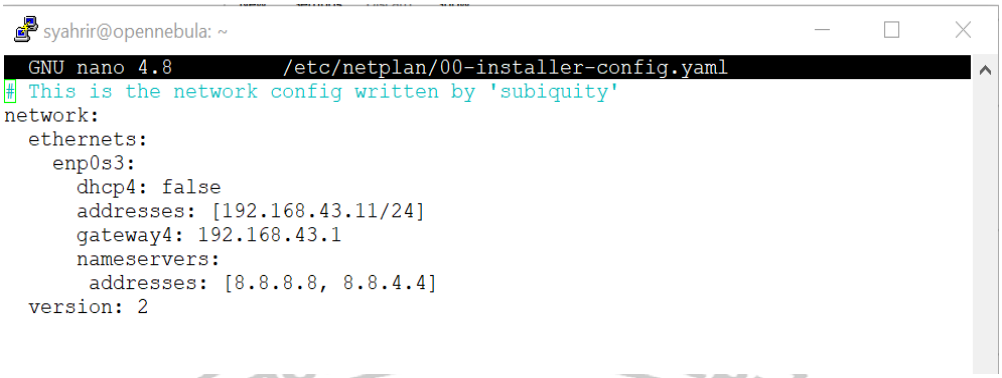


Gambar 3.6 OS Ubuntu 20.01.5

Ip address yang dipakai dalam server *load balancing* adalah **192.168.43.11/24** dengan subnet mask **255.255.255.0**. setelah OS sudah terinstall dalam pemberian ip address tersebut menggunakan proses konfigurasi ip address *static*. Pemberian ip address dilakukan secara manual sesuai dengan *network* ID dan *host* ID nya. Adapun perintah dalam proses pemberian ip address secara *static* di ubuntu adalah dengan pengetikkan di terminal yaitu:

```
sudo nano /etc/netplan/00-installer-config.yaml
```

maka hasil yang ditampilan dari perintah diatas adalah seperti Gambar 3.7

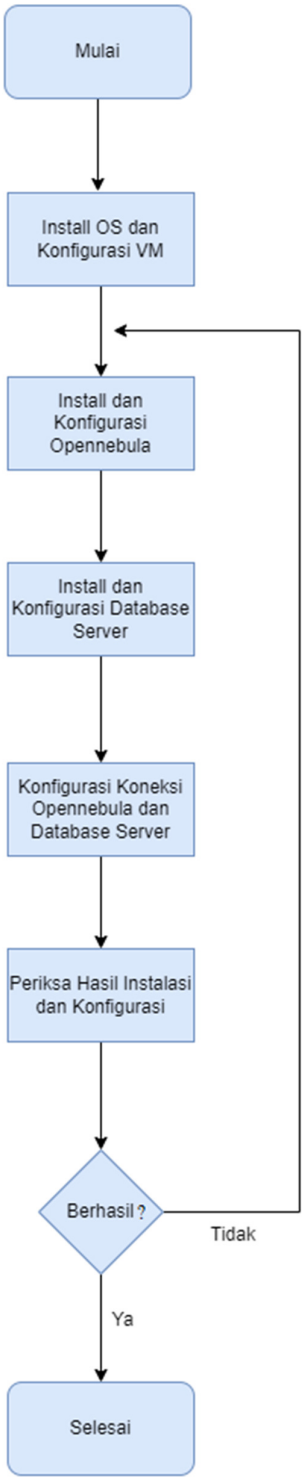


Gambar 3.7 Konfigurasi Ip Address *Static*

Kemudian setelah ip address sudah disetting maka yang harus dilakukan adalah dengan melakukan remote server menggunakan putty. Menggunakan putty jauh lebih baik karena bisa melakukan langsung konfigurasi *copy paste* jika konfigurasinya panjang. Setelah selesai dalam proses remote server selanjutnya dengan melakukan konfigurasi opennebula. Dalam proses konfigurasi yang harus dilakukan adalah dengan melakukan *wget repo key* opennebula kemudian deb sesuai dengan versi OS pada proses deb versi opennebula yang digunakan adalah 5.12 dan ubuntu 20.04

e. Flowchart Install dan Konfigurasi Koneksi Database dan Opennebula

Flowchart ini menjelaskan bagaimana cara install dan konfigurasi koneksi database dan opennebula yang dilakukan di virtualbox. untuk lebih jelas memahami cara kerja tersebut maka dibuat dengan menggunakan flowchart seperti Gambar 3.8



Gambar 3.8 Flowchart Koneksi Database

Selanjutnya pada server masing-masing node yang dimana mempunyai ip address masing-masing. Masing-masing VM dilakukan konfigurasi diterminalnya sebagai berikut

```
sudo apt install opennebula opennebula-sunstone opennebula-gate opennebula-flow
```

Melakukan instalasi dependesnya sesuai OS yang digunakan. Kemudian melakukan konfigurasi database servernya dengan menggunakan MariaDB dengan database mysql. Adapun cara konfigurasi database server yaitu dengan mengetikkan perintah

```
sudo apt -y install mariadb-server
```

Setelah selesai selanjutnya melakukan secure set beberapa *all this script* untuk lebih jelasnya dapat dilihat pada perintah dibawah ini.

```
sudo mysql_secure_installation
```

Kemudian setelah aktifkan root dan passwordnya dalam set securenya yang harus dilakukan selanjutnya yaitu dengan membuat create database opennebula. Untuk proses create database dapat dilihat pada perintah dibawah ini.

```
Sudo mysql -u root -p
CREATE DATABASE opennebula;
GRANT ALL PRIVILEGES ON opennebula.* TO ‘oneadmin’ IDENTIFIED BY ‘23mei2001’;
FLUSH PRIVILEGES;
```

```
EXIT;
```

Pada perintah diatas dengan melakukan create database artinya membuat database yang dibutuhkan. kemudian setelah itu melakukan grant *all privileges* antara database ke user, melakukan grant *all privileges* artinya mengizinkan dalam proses pembuat tabel dan hak akses oleh user atau database yang ada. Adapun nama user yang dibuat yaitu **oneadmin** dengan password **23mei2001**. Kemudian setelah mengetikan perintah *flush privileges* artinya melakukan reload ulang server dalam proses pembacaan table yang sudah dibuat setelah itu exit selesai. Untuk hasil verifikasi databasenya dapat dilihat seperti Gambar 3.9

```
MariaDB [(none)]>
MariaDB [(none)]> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| opennebula         |
| performance_schema |
+--------------------+

MariaDB [(none)]> use opennebula;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A
```

Gambar 3.9 Verifikasi Database Server

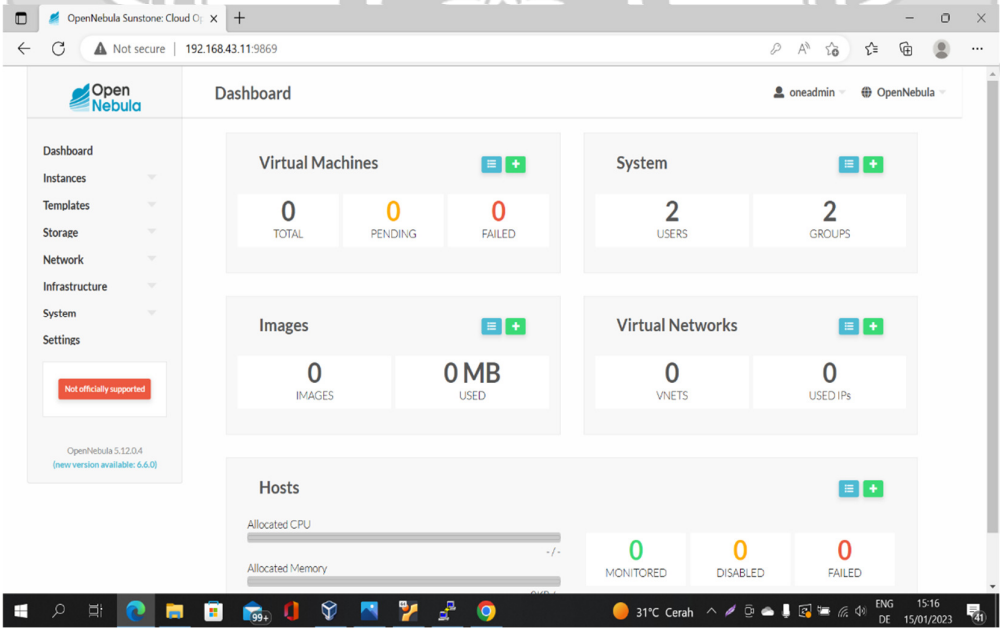
Langkah selanjutnya dengan menghubungkan antara database server yang sudah dibuat dengan opennebula yang sebelumnya sudah di install pada server opennebula. Adapun cara dalam menghubungkan database dengan server opennebula yaitu melakukan setting pada *oned* opennebula untuk masuk dalam onednya perlu mengetikan perintah di terminal yaitu *sudo nano /etc/one/oned.conf* **.** pada server isi menggunakan ip address database server yang digunakan. Lakukan

hal yang dari ke server node tersebut untuk lebih jelasnya dapat dilihat pada Gambar 3.10

Gambar 3.10 Sambungkan Database dan Server Opennebula di Oned

Merubah password pada user yang ada dalam **one_auth.** Fungsi dari merubah password adalah ketika melakukan login pada *front-end* opennebula dalam menggunakan password tersebut yang valid. Setelah itu memberikan system keamanan terhadap port yang digunakan dimana port yang digunakan yaitu **9869**. setelah semua selesai selanjutnya melakukan *start* pada *opennebula-sunstone* dan melakukan *enable*. Melakukan start dan *enable* artinya menjalankan opennebulanya sesuai dengan target *opennebula-service*. Setelah di run selanjutnya sudah bisa di akses pada browser sesuai dengan alamat ip address dan portnya yaitu **192.168.43.11:9869** maka yang ditampilkan adalah seperti Gambar 3.11 dan 3.12

Gambar 3.11 Tampilan Login OpenNebula



Gambar 3.12 Tampilan Dashboard OpenNebula.

Kemudian untuk verifikasi database pada tablenya dapat dilihat pada Gambar 3.13

```
MariaDB [opennebula]> show tables;
+----------------------------+
| Tables_in_opennebula       |
+----------------------------+
| acl                        |
| cluster_datastore_relation |
| cluster_network_relation   |
| cluster_pool               |
| cluster_vnc_bitmap         |
| datastore_pool             |
| db_versioning              |
| document_pool              |
| group_pool                 |
| group_quotas               |
| history                    |
| hook_log                   |
| hook_pool                  |
| host_monitoring            |
| host_pool                  |
| image_pool                 |
| local_db_versioning        |
| logdb                      |
| marketplace_pool           |
| marketplaceapp_pool        |
| network_pool               |
| network_vlan_bitmap        |
| pool_control               |
| secgroup_pool              |
| system_attributes          |
| template_pool              |
| user_pool                  |
| user_quotas                |
| vdc_pool                   |
| vm_import                  |
| vm_monitoring              |
| vm_pool                    |
| vm_showback                |
| vmgroup_pool               |
| vn_template_pool           |
| vrouter_pool               |
| zone_pool                  |
+----------------------------+
```

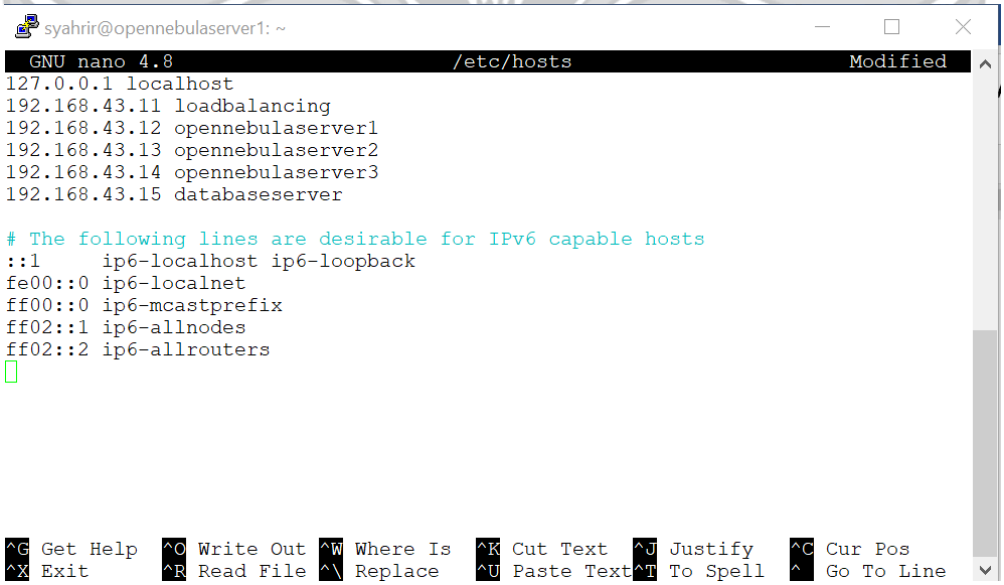
Gambar 3.13 Verifikasi Database *Table*

Mekanisme selanjutnya yaitu melakukan konfigurasi ke server node opennebula dengan langkah awal yaitu setting ip addressnya secara *static* sama

halnya yang dilakukan seperti server *load balancing* opennebula. Adapun ip address adalah

1. Opennebulaserver1: 192.168.43.12
2. Opennebulaserver2: 192.168.43.13
3. Opennebulaserver3: 192.168.43.14

Masing-masing server harus saling terkoneksi lewat hostnya sehingga semua server harus punya akses ke server lain adapun proses konfigurasi dalam menghubungkan masing-masing server seperti pada gambar 3.14. sama halnya dengan server lain baik server *load balancing* opennebula harus disetting saling menghubungkan antar satu sama lain. Dalam pengujian bahwa semua server sudah terhubung dengan menggunakan ping antar server apakah terkoneksi atau tidak. Jika belum terkoneksi coba periksa kembali masing – masing hostnya

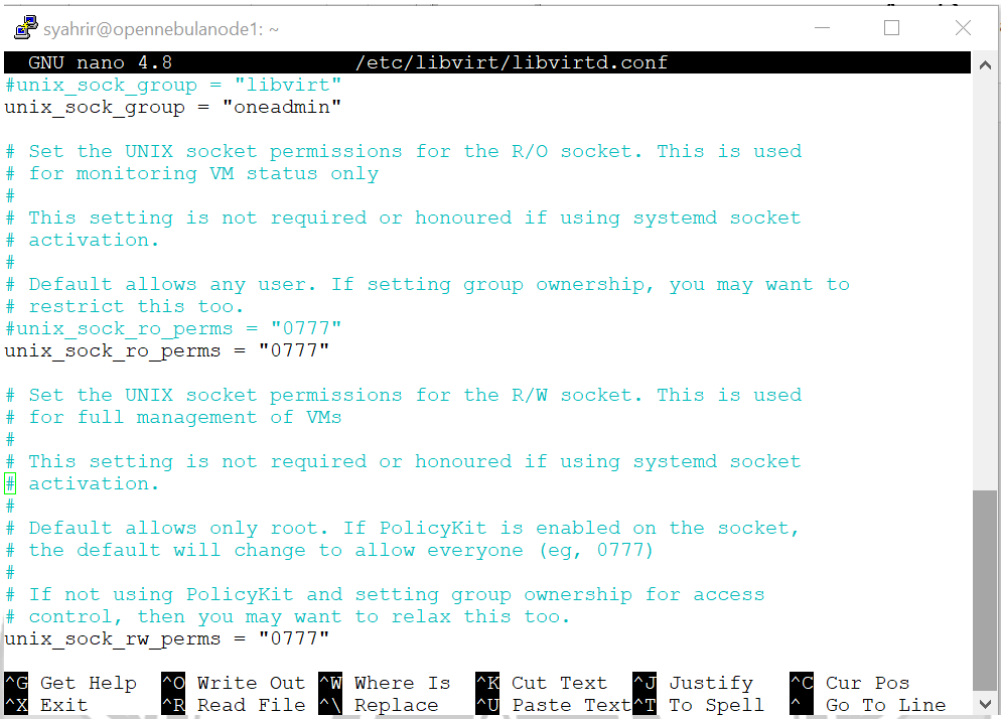


Gambar 3.14 Konfigurasi *host* masing-masing server

Setelah selesai proses konfigurasi host selanjutnya masing – masing server node harus di installkan node opennebula adapun perintah install node opennebula adalah sebagai berikut.

```
sudo apt install opennebula-node
```

Setelah terinstall nodenya ini nanti sebagai *hypervisor* antar node dan server utama dalam proses pertukaran data. Untuk melakukan proses pertukaran data perlu menggunakan jembatan yang aman ke masing-masing server sehingga harus menggunakan SSH server untuk tersambung ke masing-masing driver server node dan server utama. Sebelum melakukan konfigurasi ssh ke masing-masing server dalam masing-masing server node harus di setting ***libvirtd.conf.*** *libvirtd.conf* inilah yang akan mengelola sumber terbuka, daemon, dan alat manajemen untuk virtualisasi platform. Ini dapat digunakan untuk mengelola KVM, ESXi, QEMU. Kemudian tanpa adanya *Kernel-Based Virtual Machine* di opennebula bentuk virtualisasi tidak berfungsi kemudian sama halnya dengan QEMU. Kemudian jika QEMU ini tidak terkoneksi dengan opennebula maka tampilan VM pada server *cloud* tidak muncul untuk setting ***libvirtd.conf*** adalah seperti pada Gambar 3.15



Gambar 3.15 Setting libvirtd.conf

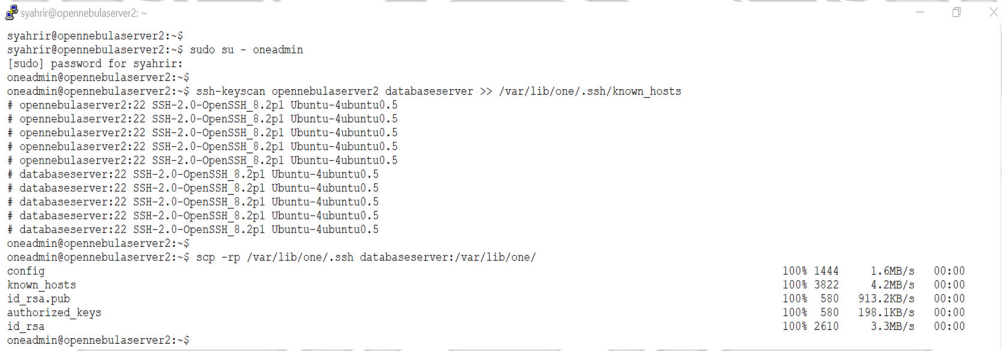
Pada Gambar 3.15 **unix_sock_group** harus disamakan dengan user databasenya agar masing-masing server node bisa menerima *resource* atau data dari server utama yaitu **oneadmin.** Kemudian **unix_sock_rw_perms** diberikan **0777** agar bisa terkendalikan tanpa terputus ke masing-masing server. Setelah selesai restart libvirtnya dengan perintah sebagai berikut.

```
sudo nano systemctl restart libvirtd

sudo nano systemctl status libvirtd
```

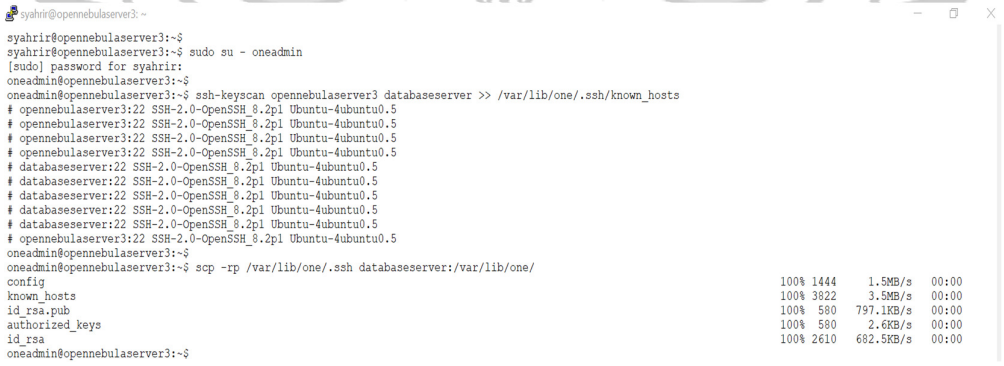
Kemudian melakukan konfigurasi mehubungkan ssh-keyscan ke masing-masing server ke dalam *known_hosts* di dalam user oneadmin. Ssh-keyscan digunakan untuk memberikan kunci yang identik di masing – masing server sehingga ketika melakukan remote ke server lain harus mempunyai kunci terlebih

dahulu tanda berhasil dalam pemberian ssh-keyscan dapat dilihat seperti pada gambar 3.16 – 3.18.

Gambar 3.16 Ssh-Keysan Node 1



Gambar 3.17 Ssh-Keysan Node 2



Gambar 3.18 Ssh-Keysan Node 3

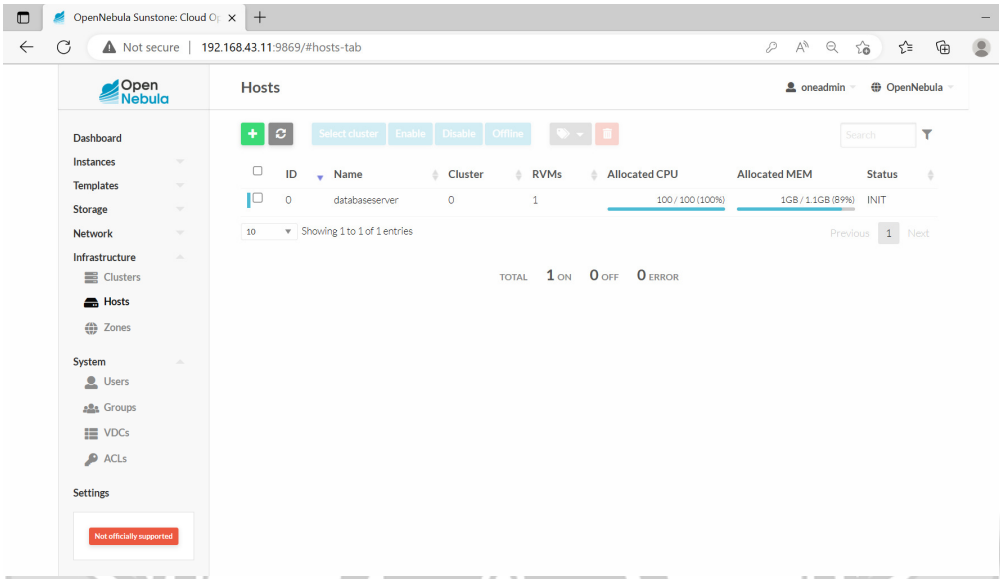
Pemberikan ssh-keysan dilakukan konfigurasi di server node opennebula. Kemudian setelah selesai jika menggunakan kunci ssh ke masing-masing server maka mengakibatkan komunikasi antar server akan terhalang dengan beberapa kali menggunakan password sehingga langkah yang digunakan adalah dengan melepas kunci pada sshnya ke masing-masing server node.

Setelah selesai coba lakukan pengujian akses ke masing-masing server dengan pengujian yang dilakukan yaitu

```
oneadmin @opennebula: ~$ ssh opennebulanode1
oneadmin @opennebulanode1: ~$ ssh opennebulanode2
oneadmin @opennebulanode2: ~$ ssh opennebulanode3
oneadmin @opennebulanode3: ~$ ssh opennebula
oneadmin @opennebula: ~$ exit
```

Setelah pengujian diatas berhasil maka langkah selanjutnya yaitu dengan memasang masing-masing node *host* pada *front end* opennebula di bagian *infrastructure* kemudian *host* dan lakukan *create host* dengan *type* yang digunakan yaitu KVM dengan *cluster* 0 atau *cluster default* setelah itu masukkan *hostname* pada database server setelah selesai maka tampilan database server dapat dilihat pada gambar 3.19. ID yang digunakan secara berurutan kemudian *name* sesuai *hostname* yang dimasukkan pada saat *create host* proses cluster yang digunakan dengan id 0 kemudian *allocated* CPU 100 di masing server node kemudian *allocated MEM* 1.1 GB dengan status masing-masing server adalah *ON*. Fungsi pada *host* adalah dengan melakukan proses distribusi data antar satu sama lain sama

halnya dengan server *load balancing* opennebula dengan proses pembagian beban antar server. Untuk lebih jelasnya dapat dilihat pada Gambar 3.19



Gambar 3.19 *Host* Database Server

f. *Flowchart* Install dan Konfigurasi *Load Balancing Haproxy*

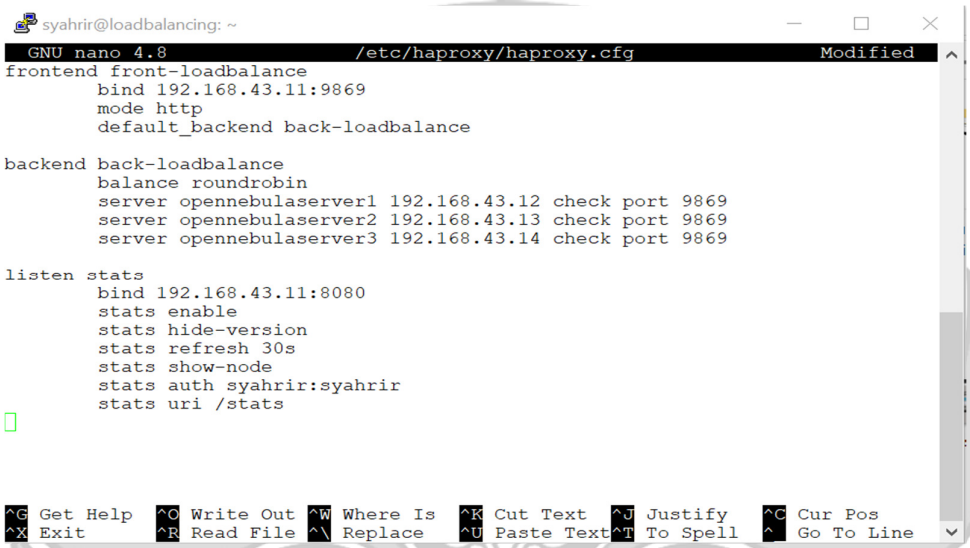
Adapun flowchartnya dapat dilihat pada Gambar 3.20

Pada Gambar 3.20 yaitu dengan melakukan instalasi *load balancing haproxy* di terminalnya dengan perintah sebagai berikut.

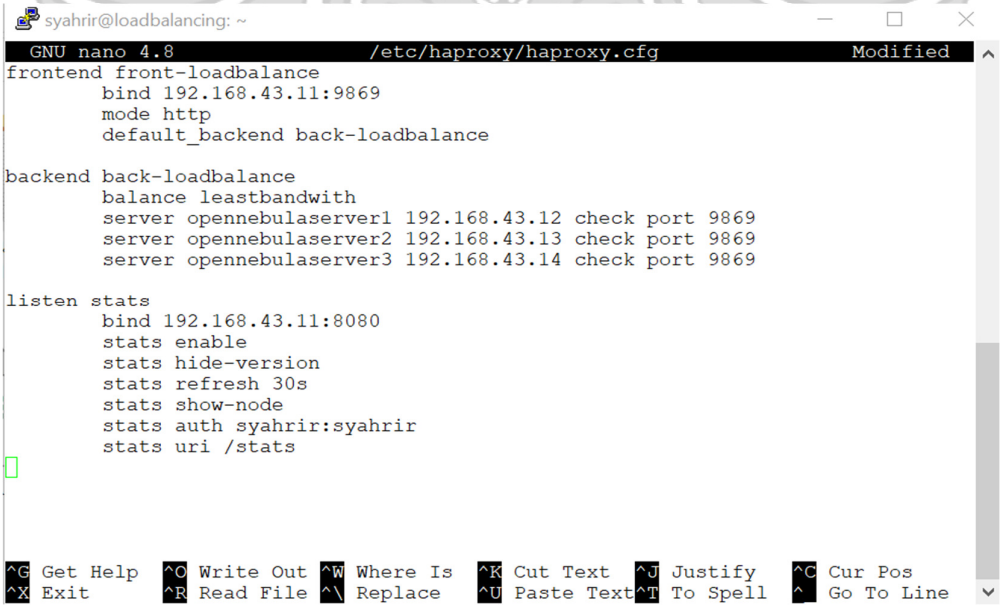
```
sudo apt install haproxy
```

Gambar 3.20 *Flowchart* Install dan Konfigurasi *Load Balancing*

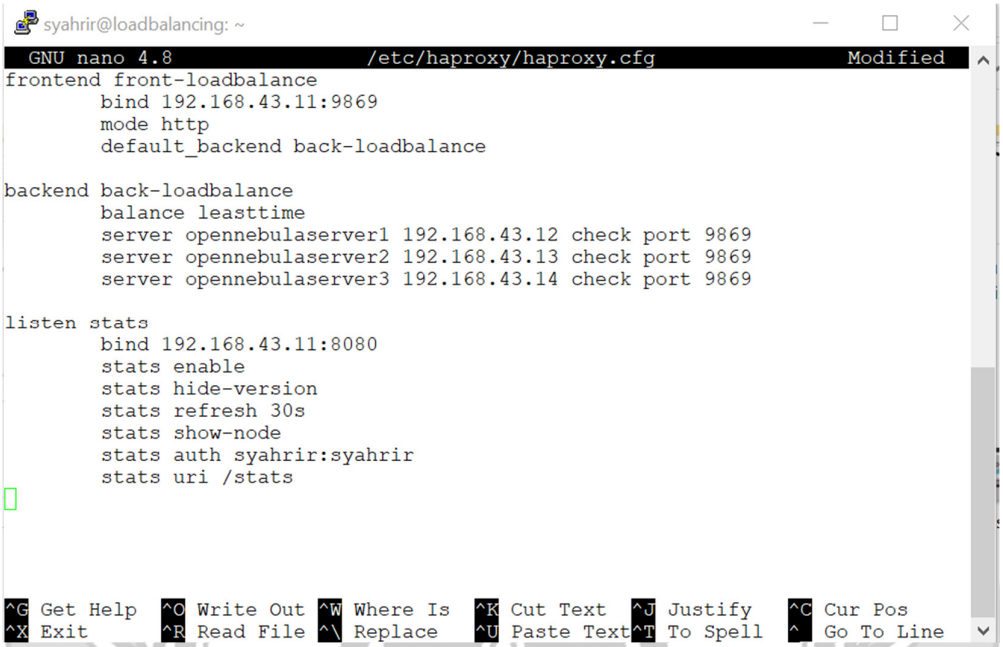
Kemudian setelah selesai install *haproxy* selanjutnya lakukan konfigurasi di dalam */etc/haproxy/haproxy.cfg* dengan menerapkan sesuai dengan algoritmanya. Adapun algoritma yang digunakan yaitu *Round robin* pada Gambar 3.21, *Least Bandwith* 3.22 dan *Least Response time* 3.23



Gambar 3.21 Konfigurasi Algoritma *Round Robin*



Gambar 3.22 Konfigurasi Algoritma *Least Bandwith*



Gambar 3.23 Konfigurasi Algoritma *Least Response Time*

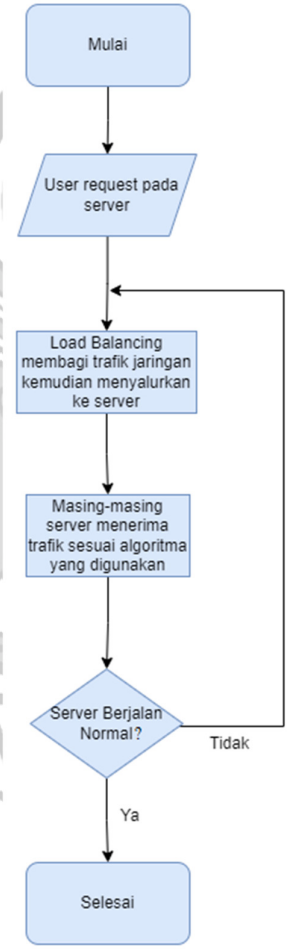
Dapat dilihat pada Gambar 3.21 – 3.23 merupakan konfigurasi penerapan algoritma *load balancing haproxy* sesuai dengan algoritma yang digunakan. Pertama definisikan front-endnya dengan akses server **192.168.122.11:9869** menggunakan port 9869 kemudian mode yang digunakan yaitu http selanjutnya dilakukan *forwardfor* ke *backend* servernya. Pada *front-end* tersebut di sinilah proses penerimaan trafik *request* HTTP sehingga trafik diarahkan ke *back-end* server di terapkan algoritma untuk pembagian beban ke masing-masing server. Dalam *listen stats* dengan port 8080 artinya jika ingin melihat algoritma berjalan silahkan lakukan pematauan menggunakan port tersebut. Setelah selesai konfigurasi lakukan verifikasi bahwa konfigurasi berhasil dengan mengetikkan perintah diterminal yaitu seperti berikut:

```
systemctl restart haproxy.service

haproxy -c -f /etc/haproxy/haproxy.cfg
```

Setelah mengetikkan perintahya dan muncul ***configuration file is valid*** itu artinya konfigurasi *load balancing haproxy* dalam penerapan algoritmanya sudah berhasil

g. Flowchart Sistem Bekerja

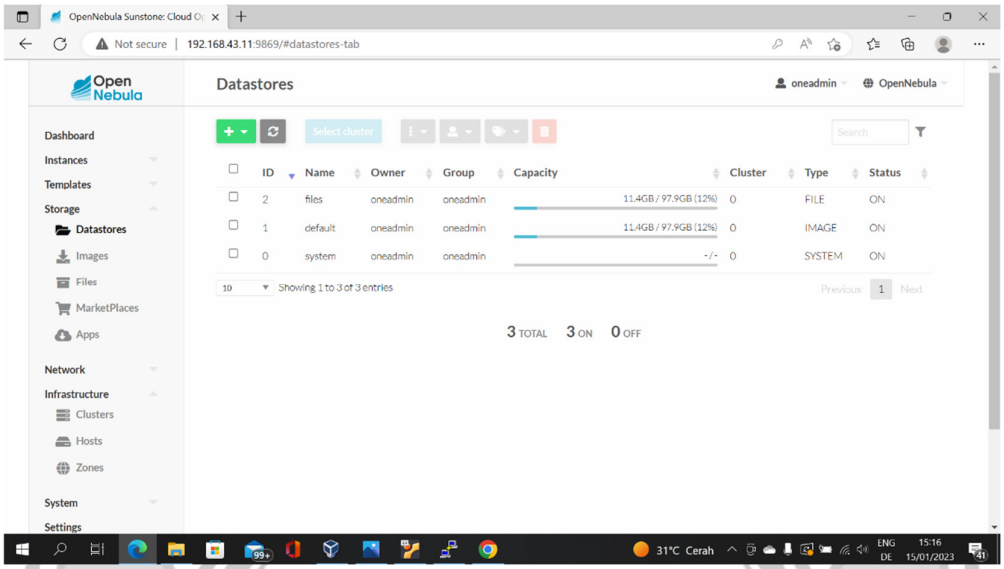
Adapun flowchart dapat dilihat pada Gambar 3.24



Gambar 3.24 *Flowchart* Sistem Bekerja

h. Pembuat Server di *Cloud Computing Opennebula*

Mekanisme dalam pembuatan server di *cloud computing* opennebula ada beberapa faktor yang harus diperhatikan seperti pastikan KVM, *enable virtualization* dan QEMU berfungsi dengan baik karena ini akan mempengaruhi dalam proses pembuatan server di *cloud* Ketika ke 3 faktor diatas tidak berfungsi maka server tidak bisa build atau *ERROR*, *UNKNOWN* dan bahkan VNC pada server tidak bisa terkoneksi sehingga tampilan server tidak bisa tampil jadi harus benar-benar diperhatikan factor tersebut. Kemudian sebelum pembuatan VM terlebih dahulu adalah harus memperhatikan *DATASTORE* pada server opennebula. *DATASTORE* dalam opennebula adalah media untuk menyimpan disk images dari virtual machine. OpenNebula mempunya datastore khusus yang digunakan untuk menyimpan disk images pada VM (*Virtual Machine*) yang berjalan yang disebut dengan *system datastore*. Datastore ini capasitynya tergantung dari server utama dan server node *hardisk* dari server node adalah 100 GB sehingga akan mengikut pada *capacitynya* di *datastore* baik dari segi files dan defaultnya dengan *cluster* 0 yang sudah dilakukan pada mekanisme sebelumnya. Untuk *owner* dan group nya sendiri menggunakan user database yang digunakan yaitu oneadmin. Data yang sudah terisi dalam penyimpanan *image* sebanyak 1 VM adalah 11,4 GB atau jika dipersenkan maka nilai persenannya senilai 12%. Capasitynya akan lebih meningkat tergantu banyaknya image atau VM yang dibuat. Adapun tampilan datastorenya seperti pada Gambar 3.25



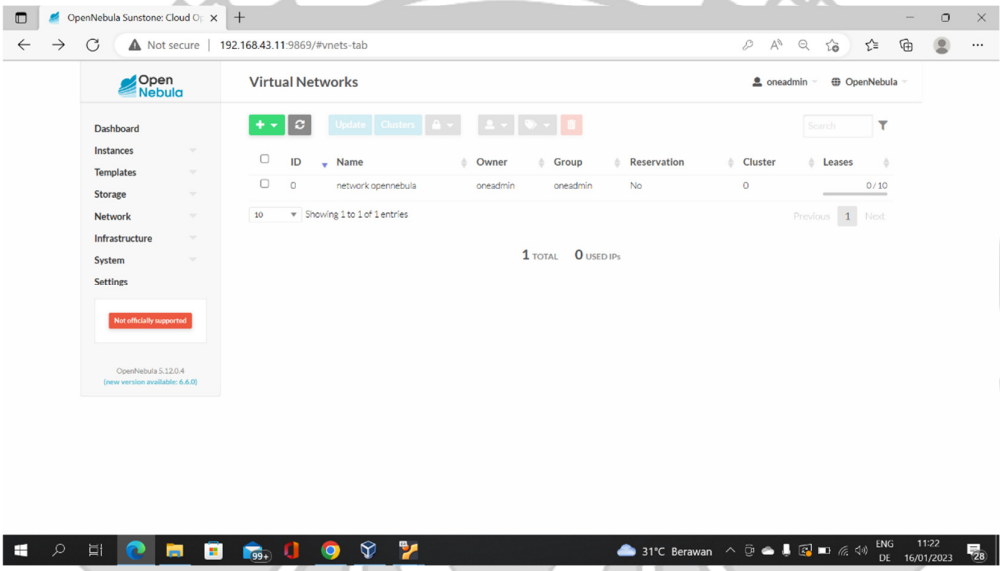
Gambar 3.25 Datastore OpenNebula

Kemudian setelah selesai yang harus dilakukan adalah dengan membuat jaringan virtual yang digunakan pada masing-masing server virtualnya. Jaringan virtual yang digunakan yaitu menggunakan metode *bridge*. Jaringan pada perangkat akan sama *network* ID pada jaringan virtual yang membedakan hanya *host* ID nya tergantung settingan staticnya. Dalam proses pembuatan jaringan virtualnya perlu dilakukan konfigurasi pada masing-masing server node. Adapun perintah yang digunakan yaitu:

```
sudo apt-get install bridge-utils
```

Menginstall *bridge-utils* artinya memberikan fungsi jaringan di server node menjadi dalam bentuk bridge. Setelah itu lanjut ke pemberian ip address pada jaringan virtual yang dilakukan pada *front end* opennebula dengan mengklik *network* kemudian *create virtual network*. Tepatnya pada bagian general isi *name* yang digunakan dalam hal ini *name virtual network* adalah **network-opennebula**

untuk clusternya sendiri adalah 0 atau *default*. Setelah itu berikan name bridgenya yaitu **virbr0** dengan network mode bridged. Untuk *physical device*nya adalah **enp0s3** kemudian first ipv4 addressnya adalah **192.168.43.11** dengan jumlah network **10** untuk lebih jelasnya mengenai jaringan virtual yang sudah dibuat dapat dilihat pada Gambar 3.26



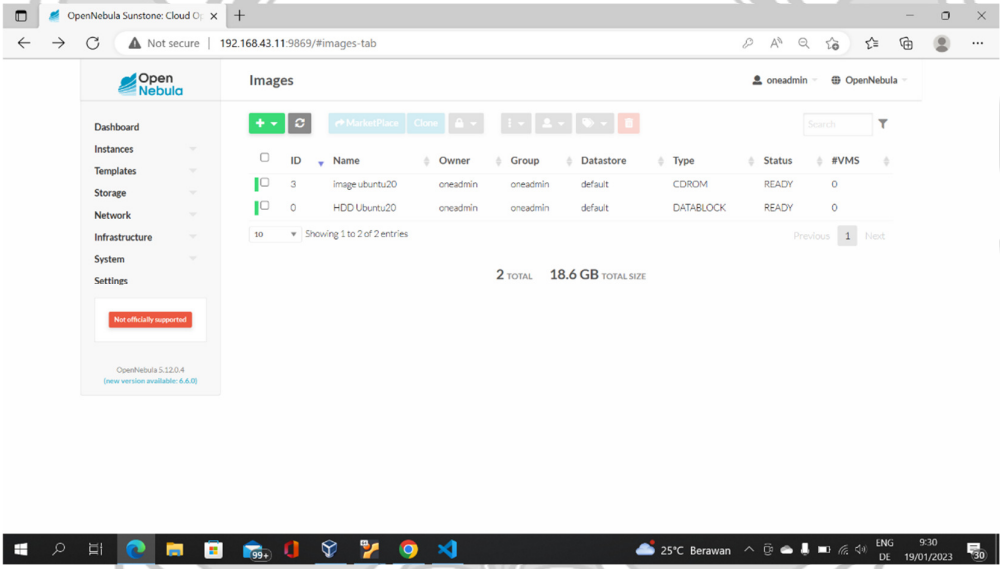
Gambar 3.26 Virtual Network OpenNebula

Mekanisme selanjutnya adalah dengan melakukan *create datablock* dan *image* pada *storage* opennebula. Dalam proses *create datablock* dan *image* ini digunakan sebagai HDD dan *image virtual* yang dapat digunakan pada saat membuat VM. langkah pertama berikan *name* HDD dan *image* nya kemudian untuk *type* yang pertama yaitu *Generic storage datablock* atau tempat penyimpanan kosong dimana isi disknya disetting 15 GB pada server yang dibuat di opennebula, kemudian *mapping drive* nya type *qcom2* menggunakan *qcom2* karena 'asli' dan

fleksibel. Yang mana support copy on *write, enkripsi, kompresi*, dan VM *shapshots*. Kemudian untuk image sendiri berikan namenya kemudian type nya berbentuk *readonly CD-ROM* dengan path URL yang digunakan dalam OS pada VM di opennebula adalah sebagai berikut :

http://repo.ugm.ac.id/iso/ubuntu/releases/focal/ubuntu-20.04.5-live-server-amd64.iso

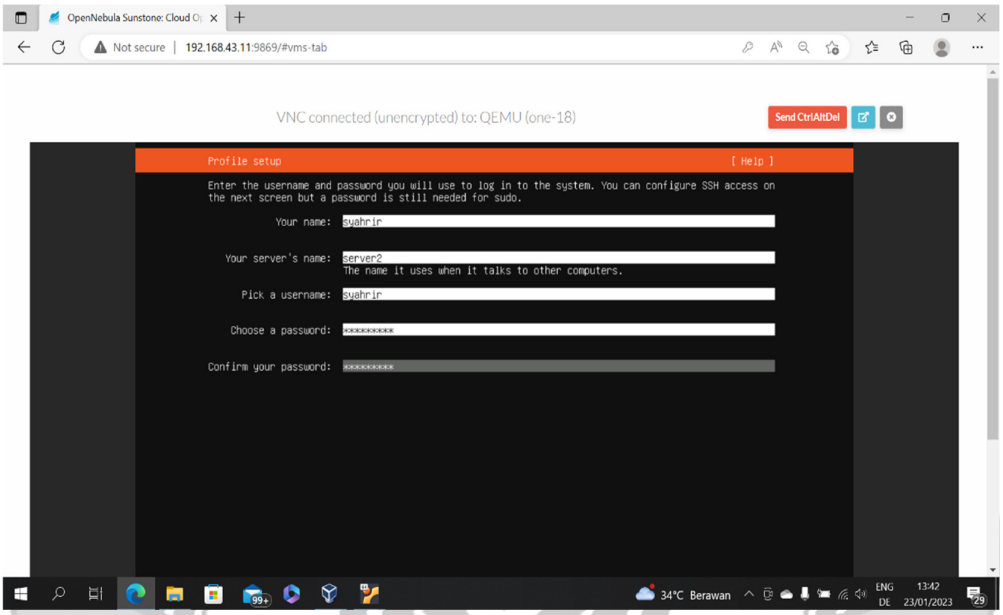
Adapun jumlah *size image* ubuntu server adalah 1.3 GB untuk hasil lebih jelasnya dapat dilihat pada Gambar 3.27



Gambar 3.27 HDD dan *Image* OpenNebula

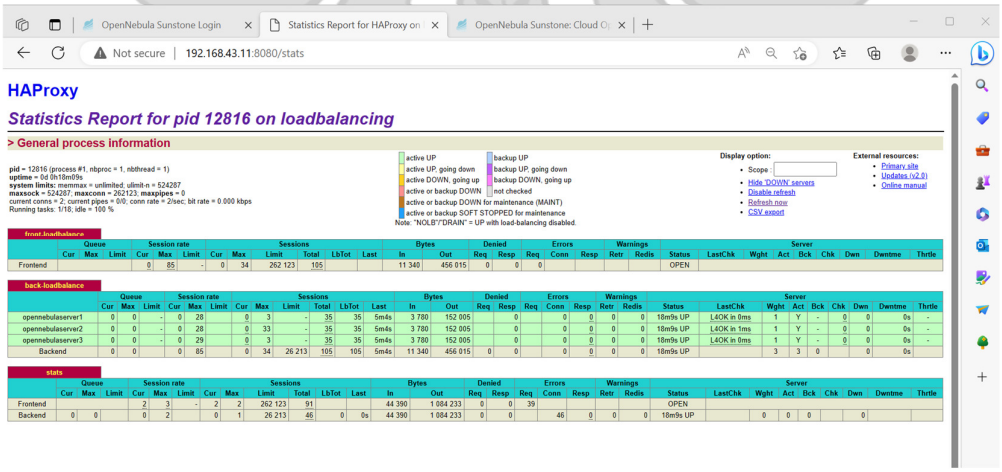
Setelah selesai langkah selanjutnya adalah dengan membuat virtual machinenya pada opennebula dibagian *instance* kemudian VMs lakukan *create* VMs. Setelah selesai dalam pembuatan server untuk masing-masing server silahkan ON kan server kemudian klik VNC nya dan lakukan instalasi ubuntu server seperti

biasanya. Untuk tampilan VNC pada server tersebut adalah seperti pada Gambar 3.28



Gambar 3.28 Tampilan VNC *connected*

Setelah selesai membuat server *cloud*, selanjutnya untuk memastikan algoritma berjalan maka silahkan akses statistic *haproxy*. Untuk lebih jelas dapat dilihat pada Gambar 3.29



Gambar 3.29 Statistik Haproxy Algoritma Berjalan

3.3.4 Implementasi

Pada tahap ini terlebih dahulu dilakukan persiapan alat dan bahan baik *hardware* maupun *software* untuk menunjang proses pembuatan sistem. Persiapan *hardware* dilakukan dengan pengadaan perangkat seperti laptop dan PC. Sedangkan persiapan *software* dilakukan dengan mempersiapkan *software* seperti sistem Operasi Windows 10, Debian 9, Haproxy dan OpenNebula. Setelah alat dan bahan tersedia tahap selanjutnya ialah dengan memulai membangun sistem

3.3.5 Pengujian

Pada tahap pengujian ini dilakukan agar bisa memastikan *Load balancing haproxy* pada OpenNebula berjalan dengan baik sesuai dengan diharapkan. Ada beberapa yang perlu di uji baik tingkat *performance* dan metode *failover* dengan melihat parameter yaitu *Latency/Response Time, Error Rate,* dan *Throughput.* Pengujian ini menggunakan Apache Jmeter jadi segala bentuk hasil request dari jmeter akan tampil dalam dashboard jmeter tersebut. Adapun Penjelasan dari parameter pengujiannya berdasarkan standar TIPHON adalah sebagai berikut :

a. *Latency/Response Time* (ms)

Latency/Response Time adalah waktu yang dibutuhkan untuk memproses permintaan dan mengirimkan respons atau parameter ini merupakan waktu minimum yang dibutuhkan dari sebuah sistem yang menerapkan algoritma penyeimbangan beban tertentu untuk merespon. Semakin kecil nilai waktu respon maka semakin bagus pula kinerja *system*.

b. *Packet Loss/Error Rate* (%)

Error Rate adalah persentase permintaan yang gagal diterima oleh server atau parameter yang menunjukkan jumlah koneksi yang tidak berhasil dibuat karena gagal dieksekusi atau diproses oleh server backend dan direpresentasikan dalam persen. *Persentase error* dapat dilihat pada kolom *Error* pada tab Aggregate Report dari *software* benchmark Apache JMeter.

c. *Throughput* (*Kbps*)

Throughput adalah parameter yang merupakan jumlah keseluruhan permintaan yang telah dieksekusi oleh sistem atau jumlah data yang diterima dan dikirimkan per detik. Parameter ini mencerminkan kapasitas server dalam hal seberapa banyak beban yang dapat ditangani. Sistem membutuhkan nilai throughput tinggi untuk mencapai kinerja yang lebih baik. Throughput diukur sebagai jumlah total transaksi dalam waktu tertentu atau transaksi per detik (TPS).

BAB IV HASIL DAN PEMBAHASAN

Hasil dari penelitian ini berupa penerapan dan pengujian metode *load balancing haproxy* pada opennebula dimana menerapkan algoritma *Least Bandwith* dan *Least Response Time*. Kemudian adapun skala pembanding yaitu menggunakan algoritma default dari *load balancing* yaitu *Round Robin*. Adapun analisis penerapan dan pengujian *load balancing haproxy* pada opennebula adalah sebagai berikut

4.1 Analisis Hasil Pengujian Metode *Load Balancing Haproxy* pada OpenNebula

Analisis pada hasil pengujian *load balancing haproxy* pada opennebula dilakukan dengan menggunakan jmeter dengan metode *thread testing*. *Thread testing* yaitu melakukan pengujian dengan melakukan permintaan HTTP Request tergantung berapa user yang ditentukan. Dalam pengujian ini pengujian dari segi perbandingan dengan parameter yang ada. Adapun bentuk perbandingannya adalah sebagai berikut :

4.1.1 Skenario 1 (Pengujian server tanpa *load balancing*)

Pada hasil pengujian *load balancing haproxy* pada opennebula dilakukan dengan pengukuran tanpa menggunakan proses pembagian beban atau tanpa *load balancing*. Pengukuran awal ini dilakukan agar supaya ada bentuk parameter awal Ketika hanya menggunakan 1 server berapa user yang bisa akses server sampai terjadi *overload.* Kemudian untuk hasil terperinci dapat dilihat pada Tabel 4.1

Tabel 4.1 *Rata-rata Latency/Response Time, Throughput dan Error Rate*

User	Server 1		
	Throughput (Kbps)	Error Rate (%)	Response Time (Ms)
100	100,5	0	3
300	331,9	0	8,5
600	483,1	0	9,1
900	592,5	0	73,23
1200	595	0	91,21
1500	598,6	0	99,32
1800	632,5	0	132,51
2100	798,4	0	141,34
2400	854,7	0	153,33
2700	968,5	0	158,45
3000	956	0,67	212,26
3300	921,6	4,75	225,77
3600	790	5,11	267,19
3900	754,7	7,43	275,32
4200	699	8,12	286,31

Berdasarkan Tabel 4.1 dapat dilihat bahwa rata-rata *throughput* ketika user yang akses sebanyak 100 yaitu 100,5 kbps dan ketika 300 user *throghput*nya bertambah hingga menjadi 331,9 kbps sehingga *interval* mencapai 231,4 kbps ketika pengukuran user hingga 4200 maka mencapai 699 kbps. Kemudian untuk *error rate* 100-2700 user masih 0 % ketika 3000-4200 sudah terjadi *error rate* pada saat 3000 user tingkat errornya masih sedikit yaitu 0,67 % tetapi ketika 4200 user naik drastis hingga mencapai 8,12 % ini diakibatkan karena banyaknya user yang akses sedangkan *resource* dari server kurang sehingga mengalami *error*. Kemudian untuk *response time* meningkat seiring banyak user yang akses hingga mencapai 4200 user maka response timenya yaitu 286,31 ms.

Hasil parameter QoS pada kondisi server 1 tanpa menggunakan *load balancing* dapat dilihat pada Tabel 4.2

Tabel 4.2 Hasil parameter QoS Server tanpa *Load Balancing*

Parameter QoS	**Users**	**Nilai**	**Standar TIPHON**	**Indeks**	**Keterangan**
Throughput (bps)	2700	968,5 Kbps	100 bps	4	Sangat Bagus
	4200	699 Kbps	100 bps	4	Sangat Bagus
Packet Loss (%)	2700	0 %	0 %	4	Sangat Bagus
	4200	8,12 %	3 %	3	Baik
Response Time (ms)	2700	158,45 ms	> 150 – 300 ms	3	Baik
	4200	286,31 ms	> 150 – 300 ms	3	Baik

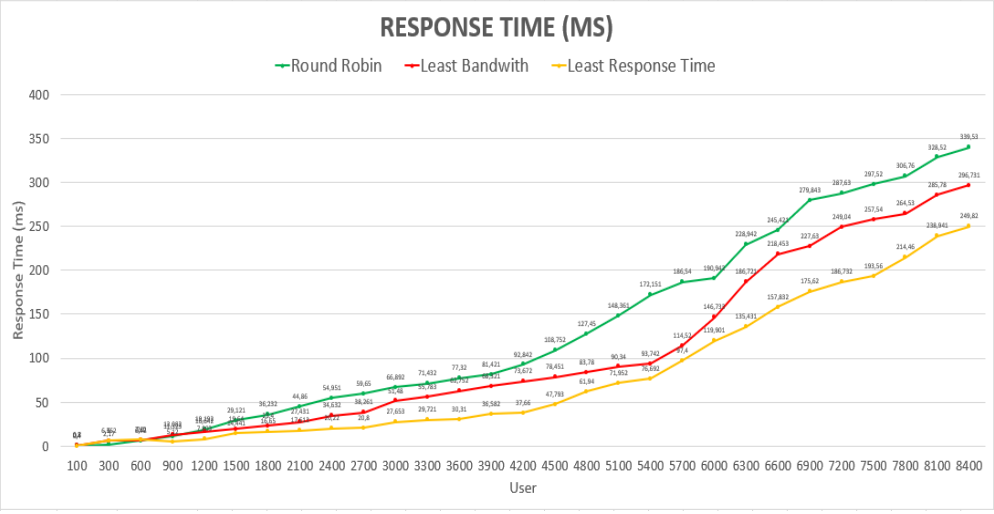
Hasil parameter QoS untuk data Skenario 1 server tanpa *load balancing* diperoleh nilai dengan masing-masing parameter yaitu user yang akses 2700 maka *throughput* sebesar 968,5 Kbps mendapatkan indeks 4 dengan kategori sangat bagus berdasarkan standar THIPON karena memenuhi nilai 100 bps sedangkan user 4200 *throughput*nya 699 Kbps berada pada indeks 4 sangat bagus. Selanjutnya untuk packet loss 2700 user diperoleh 0 % yang mendapatkan indeks 4 karena masuk pada nilai 0 % dengan kategori sangat bagus berdasarkan standar THIPON dan untuk user 4200 8,12 % berada pada standar 3 % dengan indeks 3 kategori baik. Selanjutnya untuk *response time* user 2700 diperoleh nilai sebesar 158,45 ms dengan kategori baik yang mendapatkan indeks 3 berdasarkan standar THIPON

karena memenuhi nilai standar > 150 – 300 ms sama halnya user yang akses 4200 diperoleh nilai *response time* 286,31 ms kategori baik.

4.1.2 Skenario 2 (Metode *Failover*)

Metode *failover* digunakan dalam proses pengujian algoritma yang digunakan dengan menjadi pembanding algoritma yaitu round *robin (default).* Dengan mematikan salah satu server maka dapat mengukur kinerja dengan server yang ada dimana pada pengujian hanya menggunakan 2 server.

Adapun pengujian pertama berdasarkan parameter *latency/response time* dimana merepresentasikan waktu yang dibutuhkan server dalam merespon. Jika waktu yang dibutuhkan kecil maka *performance* server tersebut berfungsi dengan baik. Adapun rata-rata *response time* dengan menggunakan metode *failover* adalah dapat dilihat pada Gambar 4.1 kemudian untuk lebih terperincinya dapat dilihat pada Tabel 4.3



Gambar 4.1 Grafik Rata-rata *Latency/Response Time Round Robin, Least Bandwith* dan *Least Response Time* Metode *Failover*

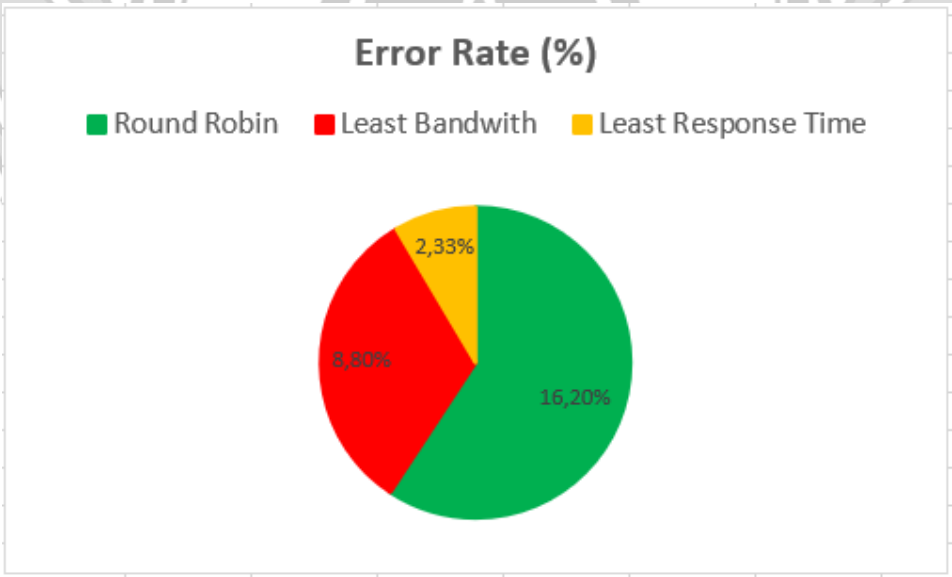
Tabel 4.3 Rata-rata *Latency/Response Time Round Robin*, *Least Bandwith* dan *Least Response Time* Metode *Failover*

User	Round Robin	Least Bandwith	Least Response Time
100	0,8	0,7	0,4
300	2,17	6,362	6
600	6,42	7,1	7,62
900	11,323	13,082	5,12
1200	19,192	16,642	7,803
1500	29,121	19,64	14,441
1800	36,232	23,4	16,65
2100	44,86	27,431	17,613
2400	54,951	34,632	20,22
2700	59,65	38,261	20,8
3000	66,892	51,48	27,653
3300	71,432	55,783	29,721
3600	77,32	62,752	30,31
3900	81,421	68,321	36,582
4200	92,842	73,672	37,66
4500	108,752	78,451	47,793
4800	127,45	83,78	61,94
5100	148,361	90,34	71,952
5400	172,151	93,742	76,692
5700	186,54	114,52	97,4
6000	190,942	146,733	119,901
6300	228,942	186,721	135,431
6600	245,421	218,453	157,832
6900	279,843	227,63	175,62
7200	287,63	249,04	186,732
7500	297,52	257,54	193,56
7800	306,76	264,53	214,46
8100	328,52	285,78	238,941
8400	339,53	296,731	249,82

Berdasarkan Gambar 4.1 dan Tabel 4.3 dapat dilihat bahwa rata-rata *Latency/Response Time* ketika user yang akses 100 maka algoritma *round robin* sebanyak 0,8 ms, *least bandwith* 0,7 ms dan *least response time* 0,4 ms. Perbandingan antara *round robin, least bandwith* dan *least response time* pada saat

batas akses user yaitu 8400 yaitu untuk *round robin* 339,53 ms, *least bandwith* 296,731 ms dan *least response time* 249,82 ms. Jadi dari ketiga algoritma tersebut dari segi response time unggul *least response time* dan *least bandwith*. Hal yang mempengaruhi meningkatnya jumlah *response time* yaitu berdasarkan dari banyaknya user yang akses secara bersamaan kemudian dari segi kapasitas server yang masing kurang sehingga mengalami peningkatan CPU dan *resource* lainnya.

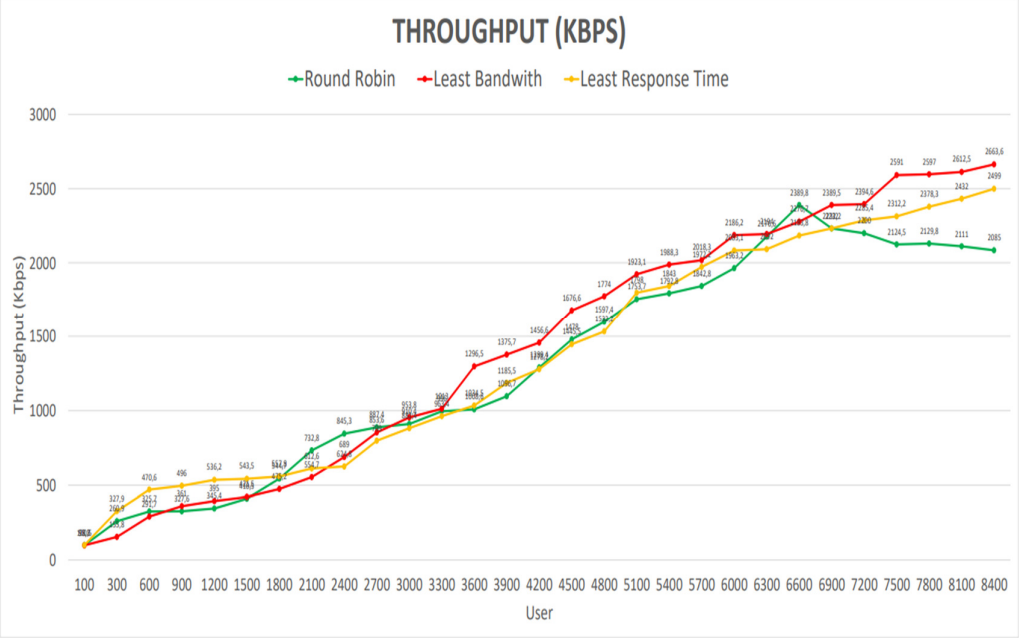
Pengujian selanjutnya yaitu *error rate/packet loss* atau tingkat *error* yang diterima oleh server atau yang gagal melakukan proses request HTTP. Dalam metode *failover* ini adapun hasil pengukuran dan pengujian berdasarkan *error rate* dapat dilihat pada Gambar 4.2.



Gambar 4.2 Diagram *Error Rate Round Robin, Least Bandwith* dan *Least Response Time* Metode *Failover*

Berdasarkan Gambar 4.2 dilihat bahwa akumulasi *error rate* hingga user 8400 adalah untuk *round robin* 16,20 %, *least bandwith* 8,80 % dan *least response time* 2,33 %. Jika dilihat dari masing-masing algoritma *error rate* tingkat rendah itu dimiliki oleh *least response time* kemudian *least bandwith.* Faktor yang mempengaruhi meningkatnya *error rate* yaitu karena banyaknya user yang akses secara bersamaan kemudian beban masing-masing server tidak mampu untuk menampung request yang ada.

Pengujian selanjutnya yaitu *throughput* dengan membandingkan dari 2 algoritma yang digunakan dan 1 algoritma jadi pembanding maka hasil pengukuran berdasarkan parameter throughputnya dapat dilihat pada Gambar 4.3 dan perincian datanya dapat dilihat pada Tabel 4.4



Gambar 4.3 Grafik *Throughput Round Robin*, *Least Bandwith* dan *Least Response Time* Metode *Failover*

Tabel 4.4 *Throughput Round Robin*, *Least Bandwith* dan *Least Response Time* Metode *Failover*

User	Round Robin	Least Bandwith	Least Response Time
100	99,8	98,7	100,5
300	260,9	155,8	327,9
600	325,7	291,7	470,6
900	327,6	361	496
1200	345,4	395	536,2
1500	410,5	421,5	543,5
1800	544,7	475,2	557,8
2100	732,8	554,7	612,6
2400	845,3	689	624,8
2700	887,4	853,6	799
3000	910,4	953,8	882,4
3300	996	1013	963,4
3600	1008,6	1296,5	1034,5
3900	1096,7	1375,7	1185,5
4200	1289,4	1456,6	1278,2
4500	1478	1676,6	1445,5
4800	1597,4	1774	1532,2
5100	1753,7	1923,1	1798
5400	1792,8	1988,3	1843
5700	1842,8	2018,3	1972,2
6000	1963,2	2186,2	2083,1
6300	2176,6	2194	2092
6600	2389,8	2276,7	2183,8
6900	2232,2	2389,5	2232
7200	2200	2394,6	2285,4
7500	2124,5	2591	2312,2
7800	2129,8	2597	2378,3
8100	2111	2612,5	2432
8400	2085	2663,6	2499

Berdasarkan Gambar 4.3 dan Tabel 4.4 dapat dilihat bahwa rata-rata *Throughput* ketika user yang akses 100 maka *round robin* mempunyai *troughput*

sebanyak 99,8 kbps, *least bandwith* 98,7 kbps dan *least response time* 100,5 kbps. Untuk *round robin* mengalami peningkatnya seiring banyak user yang request tetapi pada saat mencapai user ke 6900 throughput berkurang yaitu 2232,2 kbps dari 2389,8 kbps pada saat 6600 user sedangkan untuk algoritma least bandwith dan least response time mengalami terus peningkatan hingga mencapai batas akses request user yaitu 8400 maka throughputnya *least bandwith* yaitu 2663,6 kbps dan *least response time* 2499 kbps. Hal yang mengakibatkan throughput *round robin* menurun sedangkan *least bandwith* dan *least response time* meningkat karena berdasarkan banyaknya user yang akses dan kualitas jaringan juga berpengaruh.

Hasil parameter QoS pada kondisi skenario 2 menggunakan metode *failover* dapat dilihat pada Tabel 4.5

Tabel 4.5 Hasil parameter QoS skenario 2

Parameter QoS	**Users**	**Algoritma**	**Nilai**	**Standar TIPHON**	**Indeks**	**Keterangan**
Throughput (bps)	8400	*Round Robin*	2085 Kbps	100 bps	4	Sangat Bagus
		Least Bandwith	2663,6 Kbps	100 bps	4	Sangat Bagus
		Least Response Time	2499 Kbps	100 bps	4	Sangat Bagus
Packet Loss (%)	8400	*Round Robin*	16,20 %	15 %	2	Sedang
		Least Bandwith	8,80 %	3 %	3	Baik

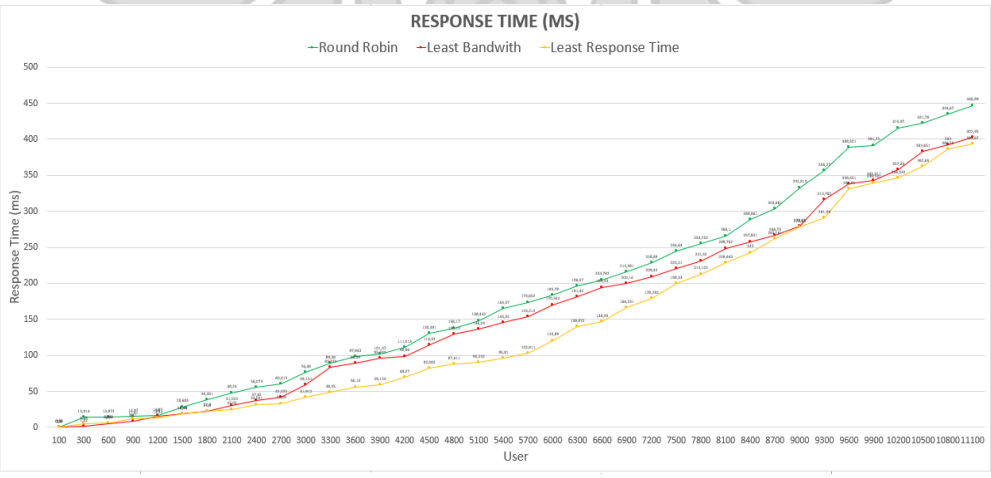
		Least Response Time	2,33 %	0 %	4	Sangat Baik
Response Time (ms)	8400	*Round Robin*	339,53 ms	> 350 – 450 ms	2	Sedang
		Least Bandwith	296,371 ms	> 150 – 300 ms	3	Baik
		Least Response Time	249,82 ms	≤ 150 ms	4	Sangat Baik

Hasil parameter QoS untuk data Skenario 2 metode *failover* pada server opennebula diperoleh nilai dengan masing-masing parameter yaitu user yang akses 8400 maka *throughput round robin* sebesar 2085 Kbps mendapatkan indeks 4 dengan kategori sangat bagus berdasarkan standar THIPON karena memenuhi nilai 100 bps kemudian algoritma *least bandwith* sebesar 2663,6 Kbps dengan kategori sangat bagus sedangkan *least response time throughput*nya 2499 Kbps berada pada indeks 4 sangat bagus. Selanjutnya untuk packet loss 8400 user diperoleh *round robin* sebanyak 16,20 % yang mendapatkan indeks 2 karena masuk pada nilai 15 % keatas dengan kategori sedang berdasarkan standar THIPON kemudian *least bandwith* sebesar 8,80 % mempunyai indek 3 yaitu baik sedangkan untuk *least response time packet loss* nya sebesar 2,33 % dengan mempunyai index 4 kategori sangat bagus. Selanjutnya untuk *response time* user 8400 dengan algoritma pembanding yaitu *round robin* sebesar 339,53 ms masuk pada standar > 350 – 450 ms dengan indeks 2 kategori sedang. Kemudian untuk algoritma *least bandwith* sebesar 296,371 ms berada pada standar > 150 – 300 ms dengan indeks 3 kategori

baik sedangkan algoritma *least response time* sebesar 249,82 ms berada pada standar TIPHON ≤ 150 ms dengan indeks 4 kategori *response time* sangat bagus.

4.1.3 Skenario 3 (Pengujian semua server menggunakan load balancing haproxy)

Pada pengujian antara *round robin*, *least bandwith* dan *least response time* dengan melihat *response time* dimulai dari dihitung saat request dikirimkan sampai request tersebut selesai. Parameter *response time* akan merepresentasikan waktu yang dibutuhkan sebuah metode *load balancing* yang mengaplikasikan algoritma *round robin, least bandwith* dan *least response time*. Semakin kecil nilai waktu respon maka kinerja *system* tersebut semakin baik. Pada *tools* Apache Jmeter, parameter *response time* dapat dilihat pada dashboard *Aggregate Report* dan *View Result in Table* dalam *average* bisa juga kolom *latency* dengan perhitungan nilai rata – rata *latency/response time* diperoleh dari total seluruh *latency/response time* dibagi dengan jumlah request. (dilihat Gambar 4.4 dan Tabel 4.6)



Gambar 4.4 Grafik Rata-rata *Latency/Response Round Robin*, *Least Bandwith* dan *Least Response Time*

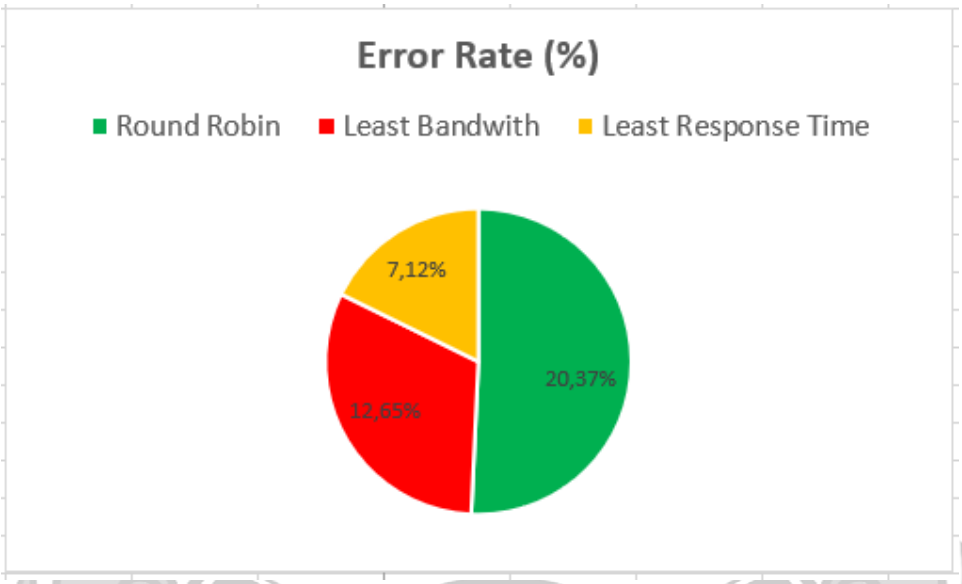
Tabel 4.6 Rata-rata *Latency/Response Time Round Robin*, *Least Bandwith* dan *Least Response Time*

User	Round Robin	Least Bandwith	Least Response Time
100	0,27	0,6	0,56
300	13,814	1,82	4,65
600	13,972	5,42	5,822
900	14,97	8,11	11,91
1200	16,87	15	13,32
1500	28,683	18,84	18,49
1800	38,301	21,9	22,9
2100	48,24	31,022	25,05
2400	56,073	37,62	32,031
2700	60,013	42,033	33,08
3000	76,48	59,151	41,912
3300	89,38	83,372	48,35
3600	97,943	88,99	56,14
3900	101,37	95,742	59,123
4200	111,013	98,86	69,27
4500	130,491	114,32	82,082
4800	138,17	129,12	87,611
5100	148,442	136,23	90,232
5400	165,37	145,32	95,81
5700	173,652	153,213	102,911
6000	183,78	170,342	120,89
6300	196,57	181,42	139,872
6600	204,782	194,53	146,93
6900	215,581	200,14	166,231
7200	228,89	209,45	179,292
7500	244,64	220,21	199,33
7800	254,752	231,32	213,122
8100	266,1	248,762	228,643
8400	288,861	257,831	243
8700	303,681	266,73	262,161
9000	332,813	279,63	278,34
9300	356,27	315,782	291,09
9600	388,501	338,431	331,05
9900	391,75	343,311	339,741
10200	415,97	357,83	346,332
10500	421,78	382,651	362,65
10800	434,67	392	386,56
11100	446,89	402,45	393,67

Berdasarkan Grafik 4.4 dan Tabel 4.6 dapat dilihat bahwa rata-rata *latency/response time* dari pengujian *load balancing* mengalami peningkatan

seiring dengan bertambahnya user. Data awal pada saat user melakukan request secara bersamaan sebanyak 100 user algoritma *round robin* mempunyai *response* yang merespon yaitu 0,27 ms, *least bandwith* 0, 6 ms sedangkan *least response time* 0,56 ms, tetapi pada saat request 11100 *round robin* meningkat drastis hingga mencapai 446,89 ms kemudian *least bandwith* 402,45 ms dan *least response time* 393,67 ms. Perbandingan antara algoritma *round robin* dan *least bandwith* berkisar 44,44 ms, kemudian algoritma *round robin* dan *least response time* berkisar 53,26 ms dan antara *least bandwith* dan *least response time* berkisar 8,78 ms. Jika dilihat dari hasil pengukuran ke 3 algoritma maka *response* yang paling sedikit adalah *least response time* jadi kinerja server meningkat jika menggunakan algoritma tersebut. Hal yang mempengaruhi ini terjadi karena banyaknya yang akses secara bersamaan sehingga baik dari resource masing-masing server akan mengalami peningkatan berdasarkan dari kapasitas server itu sendiri.

Pengujian selanjutnya yaitu parameter *error rate* yang dimana merupakan suatu hal yang dilakukan dalam memeriksa apakah system dapat menangani berbagai *error* yang mungkin terjadi di kemudian hari. Parameter ini mengacu pada jumlah *request* yang gagal atau tidak diproses oleh server atau system. Pengujian ini penting dilakukan karena dapat mengukur besar kesalahan yang dihasilkan dari algoritma yang dipakai terutama algoritma *round robin, least bandwith* dan *least response time*. Untuk tingkat presentase *error rate* dapat dilihat pada Gambar 4.5

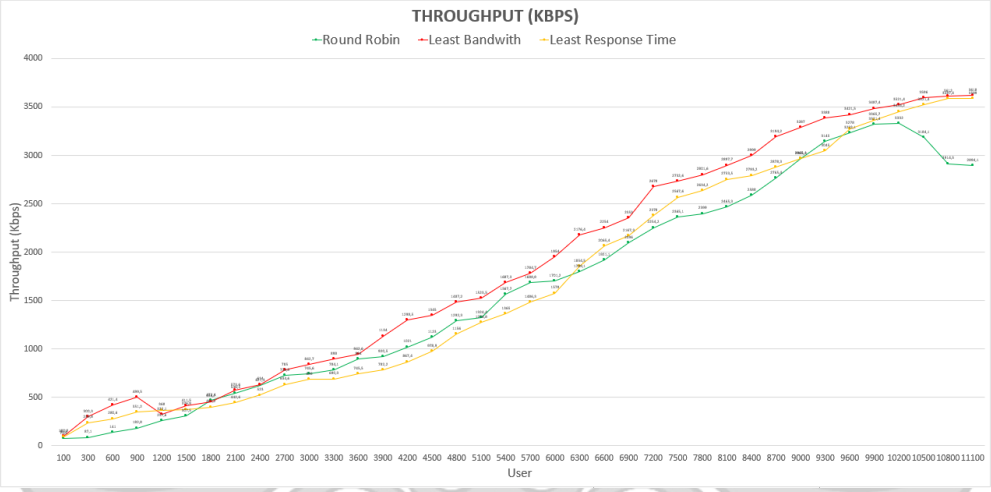


Gambar 4.5 Diagram *Error Rate Round Robin*, *Least Bandwith* dan *Least Response Time*

Berdasarkan Diagram 4.5 dapat dilihat bahwa akumulasi tingkat *error rate/packet loss* dimana presentasi dari request 11100 user *round robin* terdapat error 20,37 % kemudian untuk *least bandwith* 12,65 % dan untuk *least response time* 7,12 %. *Error* ini sering terjadi pada *protocol HTTP* disebabkan karena beberapa kondisi seperti server yang kelebihan beban karena banyak akses secara bersamaan sehingga mengakibatkan system menjadi *overload* dan dapat juga mempengaruhi *latency/response time*. Kemudian *error rate* juga berpengaruh terhadap koneksi jaringan yang tidak stabil sehingga mengisyaratkan server menunggu atau tidak mengolah apapun sehingga biasanya server akan *reboot* secara otomatis sehingga bisa di akses kembali.

Pengujian selanjutnya yaitu parameter *througput* yang dimana merupakan bentuk transaksi atau request yang dapat diproses oleh server dalam satuan waktu.

Satuan waktu yang dimaksud disini yaitu KBPS. Dalam penelitian ini, nilai throughput merepresentasikan jumlah request yang dapat diselesaikan dalam waktu 1 detik. Nilai *throughput* yang besar atau meningkat menandakan system bekerja dengan baik. Untuk skala peningkatan dari *throughput* dapat dilihat pada Grafik 4.6 dan bentuk perinciannya dapat dilihat pada Tabel 4.7



Gambar 4.6 Grafik *Throughput Round Robin, Least Bandwith* dan *Least Response Time*

Tabel 4.7 *Throughput Round Robin, Least Bandwith* dan *Least Response Time*

User	Round Robin	Least Bandwith	Least Response Time
100	75,5	100,2	91,7
300	87,1	300,3	239,8
600	141	421,4	280,6
900	182,8	499,5	351,2
1200	257,6	325,1	368
1500	307,5	411,5	376,9
1800	472,4	456,7	401,2
2100	546,5	573,6	442,6
2400	621,3	634	523
2700	728,9	785	634,6
3000	745,6	842,7	689
3300	784,1	898	690,3
3600	894	942,6	745,5
3900	920,5	1134	782,2
4200	1021	1298,5	867,4
4500	1123	1345	978,9
4800	1292,3	1487,2	1156
5100	1324,4	1523,5	1276,8
5400	1567,7	1687,3	1365
5700	1688,8	1784,7	1486,3
6000	1701,2	1954	1578
6300	1799,1	2176,4	1854,5
6600	1921,1	2254	2065,4
6900	2096	2353	2167,3
7200	2254,2	2678	2378
7500	2365,1	2732,6	2567,6
7800	2399	2801,6	2634,2
8100	2465,3	2897,7	2753,5
8400	2588	2999	2793,2
8700	2765,4	3194,2	2878,3
9000	2965,3	3287	2965,1
9300	3145	3388	3045
9600	3232,1	3421,5	3278
9900	3321,4	3487,4	3365,7
10200	3332	3521,4	3454,2
10500	3184,1	3596	3521,4
10800	2914,3	3612	3587,4
11100	2894,1	3618	3589

Berdasarkan Grafik 4.6 dan Tabel 4.7 dapat dilihat bahwa pengujian pertama yaitu 100 user dimana *throughput* yang diproses oleh *round robin* terbilang

rendah yaitu 75,5 kbps dibandingkan dengan *least bandwith* yaitu 100,2 kbps dan *least response time* yaitu 91,7 kbps. Kemudian setelah mencapai pengujian 10500 user *request round robin* mengalami penurunan *throughput* yaitu 3184,1 kbps sedangkan untuk *least bandwith* terus meningkat yaitu 3596 kbps dan *least response time* 3521,4 kbps hingga mencapai pengujian 12000 user *round robin* mempunyai nilai *throughput* 2732,4 kbps kemudian untuk *least bandwith* 3689 kbps dan *least response time* 3621,5 kbps. Jadi dari parameter *throughput* algoritma yang throughputnya banyak adalah *least bandwith*. Throughput mengalami penurunan dan peningkatan terjadi pada *backend* server yang menangani *request* menjadi kewalahan karena adanya potensi hanya 1 node yang bekerja menangani seluruh *request* dalam setiap detik.

Hasil parameter QoS pada kondisi skenario 3 menggunakan semua algoritma dan server *load balancing haproxy* opennebula Tabel 4.8

Tabel 4.8 Hasil parameter QoS skenario 3

Parameter QoS	**Users**	**Algoritma**	**Nilai**	**Standar TIPHON**	**Indeks**	**Keterangan**
Throughput (bps)	11100	*Round Robin*	2894,1 Kbps	100 bps	4	Sangat Bagus
		Least Bandwith	3618 Kbps	100 bps	4	Sangat Bagus
		Least Response Time	3589 Kbps	100 bps	4	Sangat Bagus
Packet Loss (%)	11100	*Round Robin*	20,37 %	15 %	2	Sedang

		Least Bandwith	12,65 %	3 %	3	Baik
		Least Response Time	7,12 %	3 %	3	Baik
Response Time (ms)	11100	*Round Robin*	446,89 ms	> 350 – 450 ms	2	Sedang
		Least Bandwith	402,45 ms	> 350 – 450 ms	2	Sedang
		Least Response Time	393,67 ms	> 350 – 450 ms	2	Sedang

Hasil parameter QoS untuk data Skenario 3 pada server opennebula diperoleh nilai dengan masing-masing parameter yaitu user yang akses 11100 maka *throughput round robin* sebesar 2894,1 Kbps mendapatkan indeks 4 dengan kategori sangat bagus berdasarkan standar THIPON karena memenuhi nilai 100 bps kemudian algoritma *least bandwith* sebesar 3618 Kbps dengan kategori sangat bagus sedangkan *least response time throughput*nya 3589 Kbps berada pada indeks 4 sangat bagus. Selanjutnya untuk packet loss 11100 user diperoleh *round robin* sebanyak 20,37 % yang mendapatkan indeks 2 karena masuk pada nilai 15 % keatas dengan kategori sedang berdasarkan standar THIPON kemudian *least bandwith* sebesar 12,65 % mempunyai indek 3 yaitu baik sedangkan untuk *least response time packet loss* nya sebesar 7,12 % dengan mempunyai index 3 kategori baik. Selanjutnya untuk *response time* user 11100 dengan algoritma pembanding yaitu *round robin* sebesar 446,89 ms masuk pada standar > 350 – 450 ms dengan indeks 2 kategori sedang. Kemudian untuk algoritma *least bandwith* sebesar 402,45 ms

berada pada standar > 350 – 450 ms dengan indeks 2 kategori sedang sedangkan algoritma *least response time* sebesar 393,67 ms berada pada standar TIPHON> 350 – 450 ms dengan indeks 2 kategori *response time* sedang.

BAB V KESIMPULAN DAN SARAN

5.1 Kesimpulan

Setelah melakukan penerapan metode *load balancing* haproxy pada opennebula *cloud computing* dan melakukan pengujian terhadap beberapa user secara bersamaan dalam melakukan request terhadap spesifikasi server yang sama agar mampu memberikan kinerja server dalam *cloud computing* yang lebih tinggi sehingga dapat mengurangi adanya server *overload* dan meningkatkan kinerja server yang lebih baik. Maka berdasarkan hasil penerapan dan pengujian serta hasil analisis yang dilakukan pada penelitian ini, maka diperoleh kesimpulan sebagai berikut :

1. *Load Balancing Haproxy* berhasil membagi beban secara merata ke masing-masing server opennebula sehingga mendapatkan kinerja server yang lebih baik.
2. Hasil pengukuran *load balancing* dengan *request* user 11100 dengan standar berdasarkan *Telecommunications and Internet Protocol Harmonization Over Networks (TIPHON)* dimana *response time* untuk algoritma *round robin* sebesar 446,89 ms masuk kategori sedang, algoritma *least bandwith* sebesar 402,45 ms kategori sedang dan *least response time* sebesar 393,67 ms masuk dalam kategori baik. Kemudian *packet loss/Error rate* (%) untuk algoritma *round robin* 20,37 %, *least bandwith* 12,65 % dan *least response time* 7,12 % masuk kategori baik berdasarkan standar. Sedangkan *throughput round robin* 2894,1 Kbps, *least bandwith* 3618 Kbps dan *least response time* 3589 Kbps masuk dalam indeks 4 kategori sangat baik

5.2 Saran

Adapun saran untuk pengembangan penelitian ini selanjutnya antara lain:

1. Mengembangkan algoritma yang lain untuk diterapkan pada *load balancing* haproxy pada opennebula agar supaya memperoleh kinerja server yang lebih baik
2. Menggunakan server fisik utama yang mempunyai *resource* yang besar seperti CPU, RAM dan Hardisk.
3. Menggunakan ip public sehingga server dapat dijangkau oleh semua orang yang ingin melakukan *create cloud computing* pada opennebula

DAFTAR PUSTAKA

Adani, m. R. (2020). *Apa itu mysql: pengertian, fungsi, beserta kelebihan*. Sekawanmedia.co.id.Https://www.sekawanmedia.co.id/blog/pengertian mysql/

Ajeng ayu winarsih. (2021). *Jaringan komputer, pengertian, jenis, transmisi, dan topologi*.mediaindonesia.com.https://mediaindonesia.com/teknologi/433330/jarin gan-komputerpengertian-jenis-transmisi-dan-topologi

Alankar, b., sharma, g., kaur, h., valverde, r., & chang, v. (2020). Experimental setup for investigating the efficient load balancing algorithms on virtual cloud. *Sensors*, *20*(24), 7342.

ETSI. (1999). Telecommunications and Internet Protocol Harmonization Over Networks (TIPHON); General aspects of Quality of Service (QoS). Etsi Tr 101 329 V2.1.1, 1, 1–37.

Hartawan, d. (2021). *Bagaimana cara kerja database server? Cari tahu semuanya di sini*. Toffeedev.com. Https://toffeedev.com/blog/cara-kerja-database-server/

ibrahim, i. M., ameen, s. Y., yasin, h. M., omar, n., kak, s. F., rashid, z. N., salih, a. A., salim, n. O. M., & ahmed, d. M. (2021). Web server performance improvement using dynamic load balancing techniques: a review. *System*, *19*, 21.

Imanudin, a. (2018). *Virtualisasi server berbasis proxmox ve*. Excellent publishing.

Jader, o. H., zeebaree, s. R., & zebari, r. R. (2019). A state of art survey for web server performance measurement and load balancing mechanisms. *International journal of scientific & technology research*, *8*(12), 535–543.

Khair, y., dennai, a., & elmir, y. (2021). An experimental performance evaluation of opennebula and eucalyptus cloud platform solutions. *International conference on artificial intelligence in renewable energetic systems*, 450–457.

Leman, d. (2019). Load balancing 2 jalur internet menggunakan mikrotik round robin. *Rjocs (riau journal of computer science)*, *5*(2), 137–143.

Madakatte, b. K., & nagesh, h. R. (2021). Arr and fusion based virtual machine scheduling techniques for eucalyptus cloud. *Webology (issn: 1735-188x)*, *18*(6).

Majumder, s. (2021a). *Least bandwidth method*. Citrix.com. Https://docs.citrix.com/enus/citrix-adc/current-release/load-balancing/load-balancing-customizingalgorithms/leastbandwidth-method.html

Majumder, s. (2021b). *Least response time method*. Citrix.com. Https://docs.citrix.com/en-us/citrix-adc/current-release/load-balancing/load-Balancing-customizing-algorithms/leastresponsetime-method.html

Opennebula. (n.d.). *An overview of opennebula*. Opennebula.io. Retrieved July19,2022,fromhttps://docs.opennebula.io/4.4/design_and_installation/getting_started/intro .html

Riskiono, s. D., & pasha, d. (2020). Analisis metode load balancing dalam meningkatkan kinerja website e-learning. *Jurnal teknoinfo*, *14*(1), 22–26.

Sekarputri, n. (2022). *Begini cara kerja observium untuk jaringan anda*. Sekawanmedia.co.id. Https://www.sekawanmedia.co.id/blog/apa-itu-observium/

sudirman, d. (2020). *Opennebula vs openstack, 2 layanan cloud pilihan*. Www.virtualiable.com.https://virtualiable.com/opennebula-vs-Openstack/#:~:text=opennebula telah dibentuk selama hampir satu dekade yang,mana mereka dapat menyajikan fitur pembaharuan untuk pengguna.

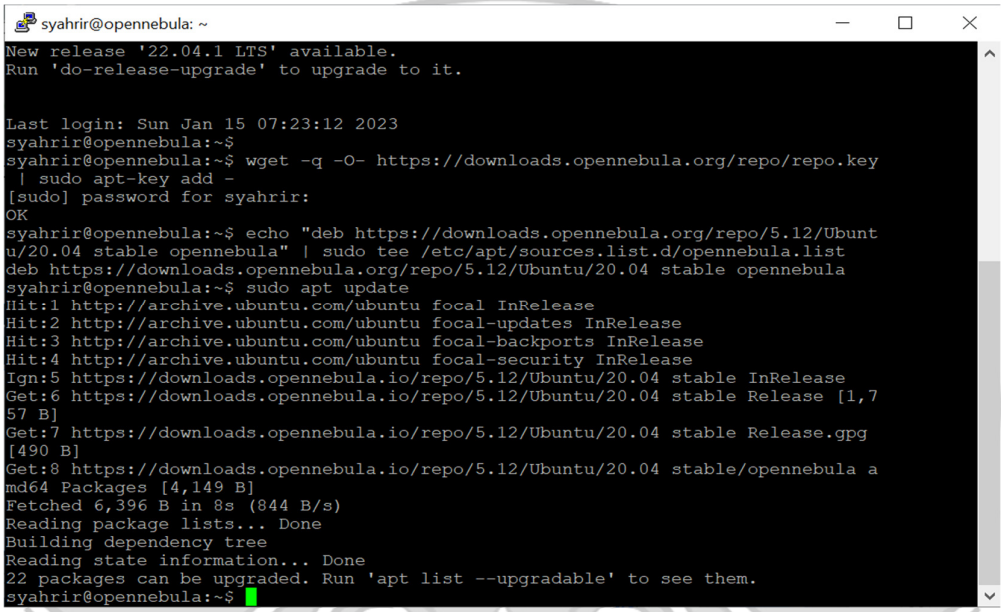
Syamsu, s. (2018). Implementasi cluster database berbasis mysql dan haproxy sebagai pembagi beban kerja server. *Inspiration: jurnal teknologi informasi dan komunikasi*, *8*(1), 48–58.

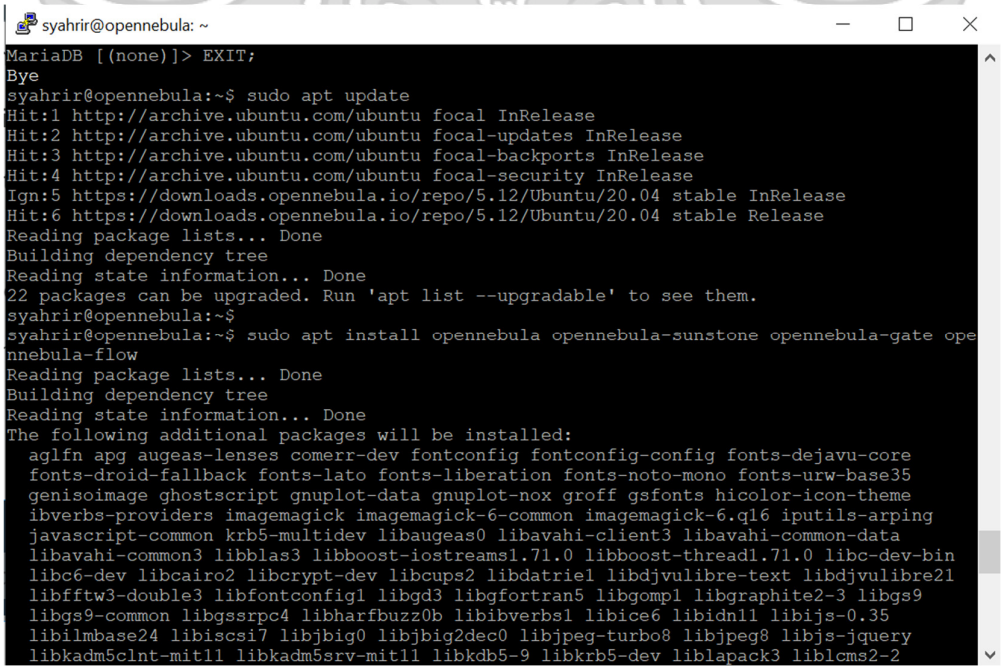
Thapliyal, n., & dimri, p. (n.d.). *Load balancing in cloud computing based on honey bee foraging behavior and load balance min-min scheduling algorithm*.

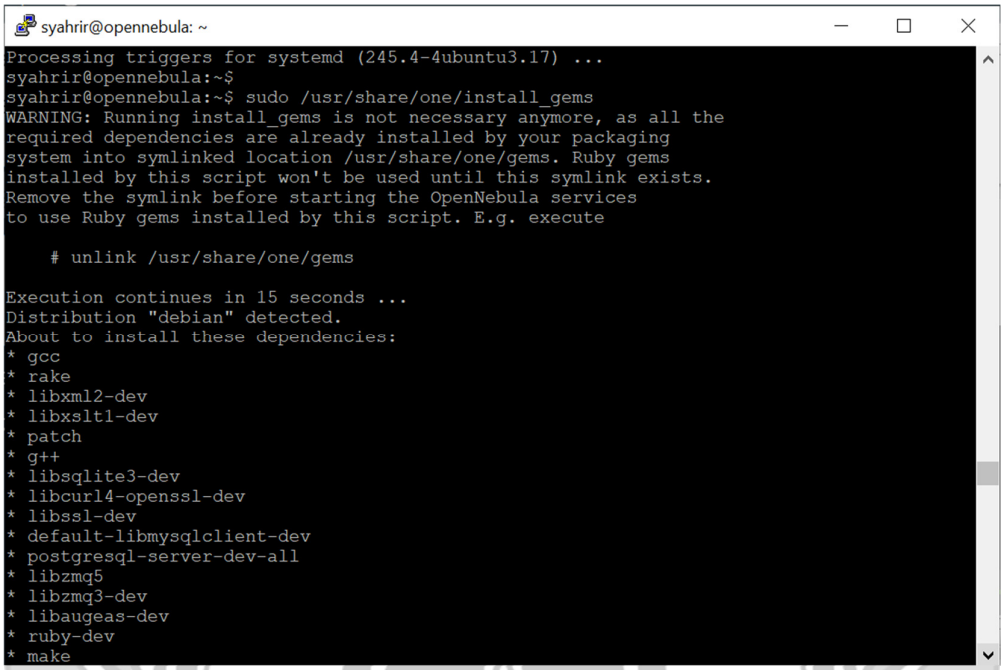
Widarma, a., & siregar, y. H. (2019). Analisis kinerja teknologi virtualisasi server (study kasus: universitas asahan). *Seminar nasional multi disiplin ilmu universitas asahan*.

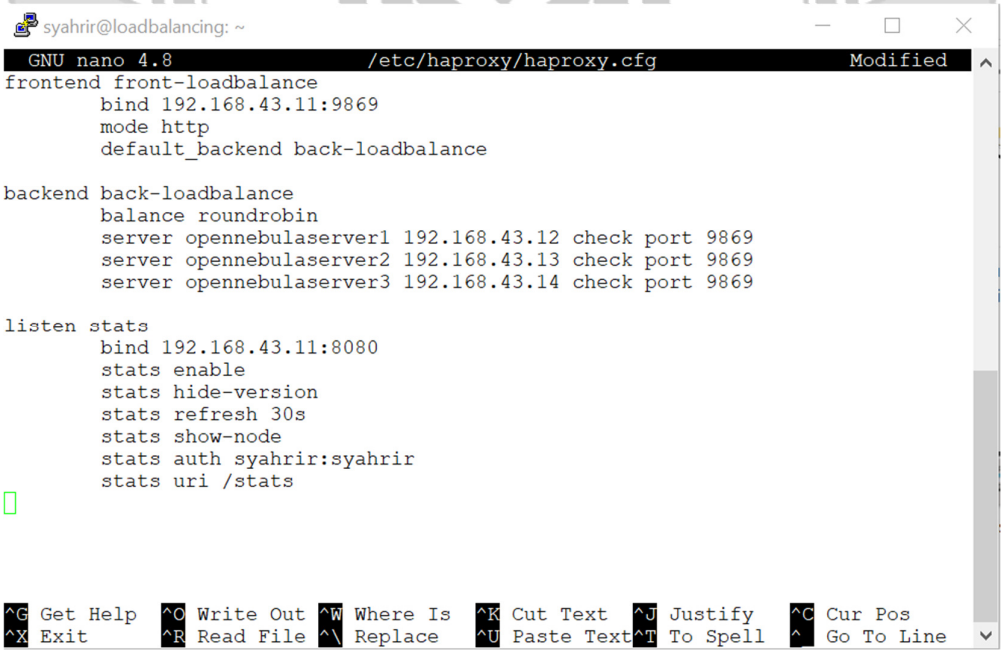
LAMPIRAN

Lampiran 1 Dokumentasi Konfigurasi *Opennebula*, Database Server dan *Load Balancing Haproxy*






syahrir@opennebula: ~
Processing triggers for systemd (245.4-4ubuntu3.17) ...
syahrir@opennebula:~$
syahrir@opennebula:~$ sudo /usr/share/one/install_gems
WARNING: Running install_gems is not necessary anymore, as all the
required dependencies are already installed by your packaging
system into symlinked location /usr/share/one/gems. Ruby gems
installed by this script won't be used until this symlink exists.
Remove the symlink before starting the OpenNebula services
to use Ruby gems installed by this script. E.g. execute
unlink /usr/share/one/gems
Execution continues in 15 seconds ...
Distribution "debian" detected.
About to install these dependencies:
* gcc
* rake
* libxml2-dev
* libxslt1-dev
* patch
* g++
* libsqlite3-dev
* libcurl4-openssl-dev
* libssl-dev
* default-libmysqlclient-dev
* postgresql-server-dev-all
* libzmq5
* libzmq3-dev
* libaugeas-dev
* ruby-dev
* make


syahrir@loadbalancing: ~
GNU nano 4.8 /etc/haproxy/haproxy.cfg Modified
frontend front-loadbalance
bind 192.168.43.11:9869
mode http
default_backend back-loadbalance
backend back-loadbalance
balance roundrobin
server opennebulaserver1 192.168.43.12 check port 9869
server opennebulaserver2 192.168.43.13 check port 9869
server opennebulaserver3 192.168.43.14 check port 9869
listen stats
bind 192.168.43.11:8080
stats enable
stats hide-version
stats refresh 30s
stats show-node
stats auth syahrir:syahrir
stats uri /stats
^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify ^C Cur Pos
^X Exit ^R Read File ^\ Replace ^U Paste Text ^T To Spell ^_ Go To Line

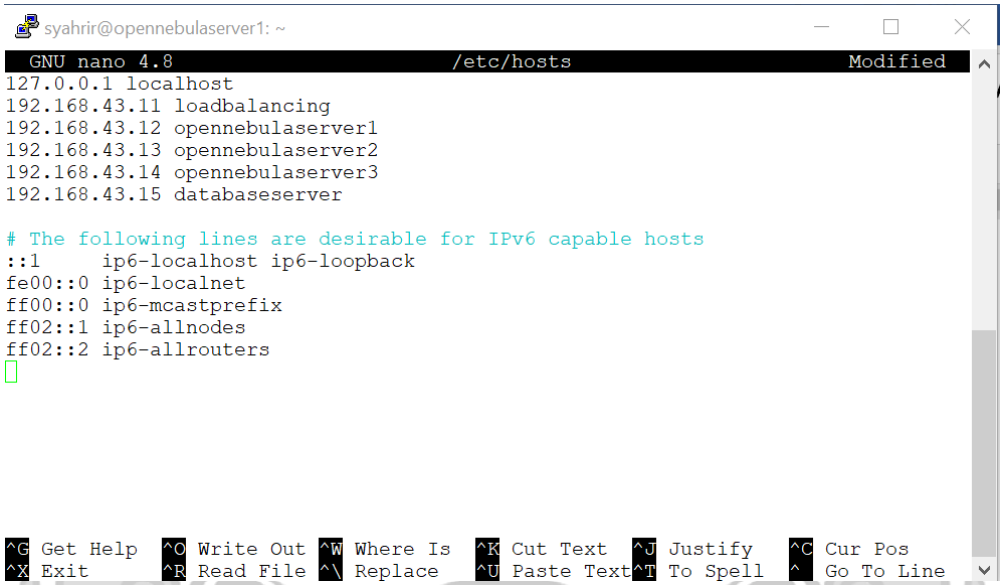
```
MariaDB [(none)]> CREATE DATABASE opennebula;
Query OK, 1 row affected (0.001 sec)

MariaDB [(none)]> GRANT ALL PRIVILEGES ON opennebula.* TO 'oneadmin' IDENTIFIED BY '23mei
2001';
Query OK, 0 rows affected (0.001 sec)

MariaDB [(none)]> FLUSH PRIVILEGES;
Query OK, 0 rows affected (0.000 sec)

MariaDB [(none)]> EXIT;
Bye
syahrir@opennebula:~$
```

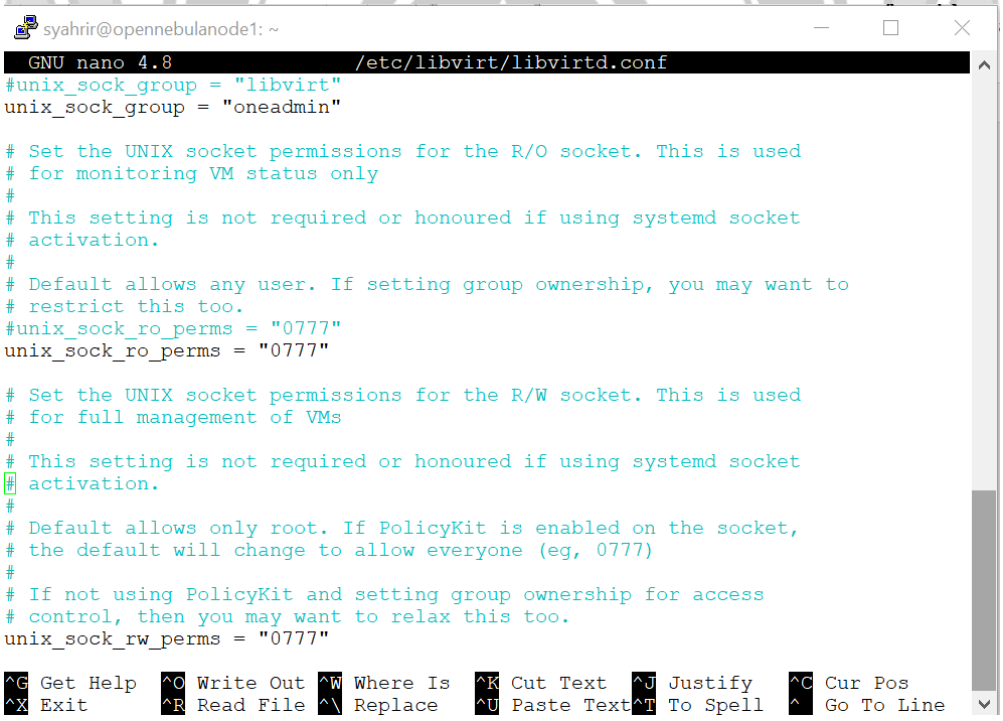
```
MariaDB [(none)]>
MariaDB [(none)]> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| opennebula         |
| performance_schema |
+--------------------+
```

```
MariaDB [(none)]> use opennebula;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A
```

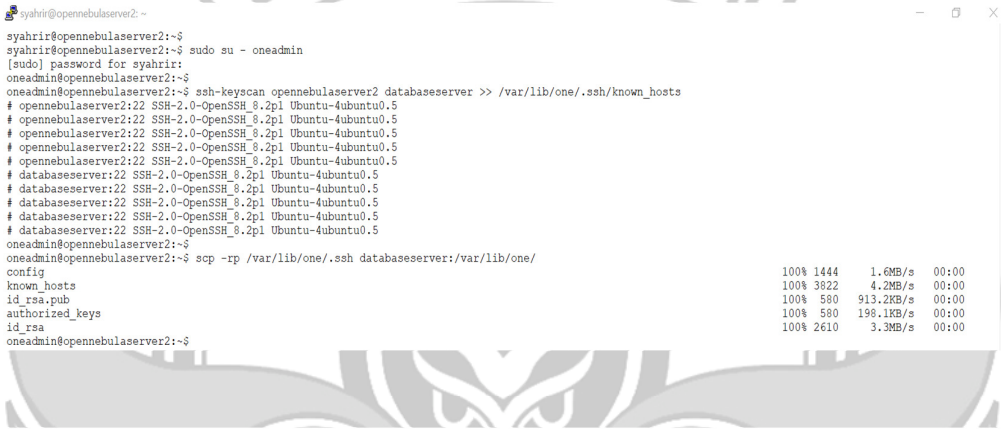
```
syahrir@opennebulaserver1: ~
 GNU nano 4.8                    /etc/one/oned.conf                 Modified


#DB = [ BACKEND = "sqlite",
#       TIMEOUT = 2500 ]

# Sample configuration for MySQL
 DB = [ BACKEND = "mysql",
        SERVER  = "192.168.43.15",
        PORT    = 0,
        USER    = "oneadmin",
        PASSWD  = "23Mei2001",
        DB_NAME = "opennebula",
        CONNECTIONS = 25,
        COMPARE_BINARY = "no" ]

VNC_PORTS = [
    START    = 5900,
    RESERVED = "32768:65536"
    # RESERVED = "6800, 6801, 6810:6820, 9869"
]

^G Get Help  ^O Write Out ^W Where Is  ^K Cut Text  ^J Justify   ^C Cur Pos
^X Exit      ^R Read File ^\ Replace   ^U Paste Text^T To Spell  ^_ Go To Line
```

```
MariaDB [opennebula]> show tables;
+----------------------------+
| Tables_in_opennebula       |
+----------------------------+
| acl                        |
| cluster_datastore_relation |
| cluster_network_relation   |
| cluster_pool               |
| cluster_vnc_bitmap         |
| datastore_pool             |
| db_versioning              |
| document_pool              |
| group_pool                 |
| group_quotas               |
| history                    |
| hook_log                   |
| hook_pool                  |
| host_monitoring            |
| host_pool                  |
| image_pool                 |
| local_db_versioning        |
| logdb                      |
| marketplace_pool           |
| marketplaceapp_pool        |
| network_pool               |
| network_vlan_bitmap        |
| pool_control               |
| secgroup_pool              |
| system_attributes          |
| template_pool              |
| user_pool                  |
| user_quotas                |
| vdc_pool                   |
| vm_import                  |
| vm_monitoring              |
| vm_pool                    |
| vm_showback                |
| vmgroup_pool               |
| vn_template_pool           |
| vrouter_pool               |
| zone_pool                  |
+----------------------------+
```


syahrir@opennebulaserver1: ~
GNU nano 4.8 /etc/hosts Modified
127.0.0.1 localhost
192.168.43.11 loadbalancing
192.168.43.12 opennebulaserver1
192.168.43.13 opennebulaserver2
192.168.43.14 opennebulaserver3
192.168.43.15 databaseserver
The following lines are desirable for IPv6 capable hosts
::1 ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify ^C Cur Pos
^X Exit ^R Read File ^\ Replace ^U Paste Text ^T To Spell ^_ Go To Line


syahrir@opennebulanode1: ~
GNU nano 4.8 /etc/libvirt/libvirtd.conf
#unix_sock_group = "libvirt"
unix_sock_group = "oneadmin"
# Set the UNIX socket permissions for the R/O socket. This is used
# for monitoring VM status only
#
This setting is not required or honoured if using systemd socket
activation.
#
Default allows any user. If setting group ownership, you may want to
restrict this too.
#unix_sock_ro_perms = "0777"
unix_sock_ro_perms = "0777"
# Set the UNIX socket permissions for the R/W socket. This is used
# for full management of VMs
#
This setting is not required or honoured if using systemd socket
activation.
#
Default allows only root. If PolicyKit is enabled on the socket,
the default will change to allow everyone (eg, 0777)
#
# If not using PolicyKit and setting group ownership for access
# control, then you may want to relax this too.
unix_sock_rw_perms = "0777"
^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify ^C Cur Pos
^X Exit ^R Read File ^\ Replace ^U Paste Text ^T To Spell ^_ Go To Line

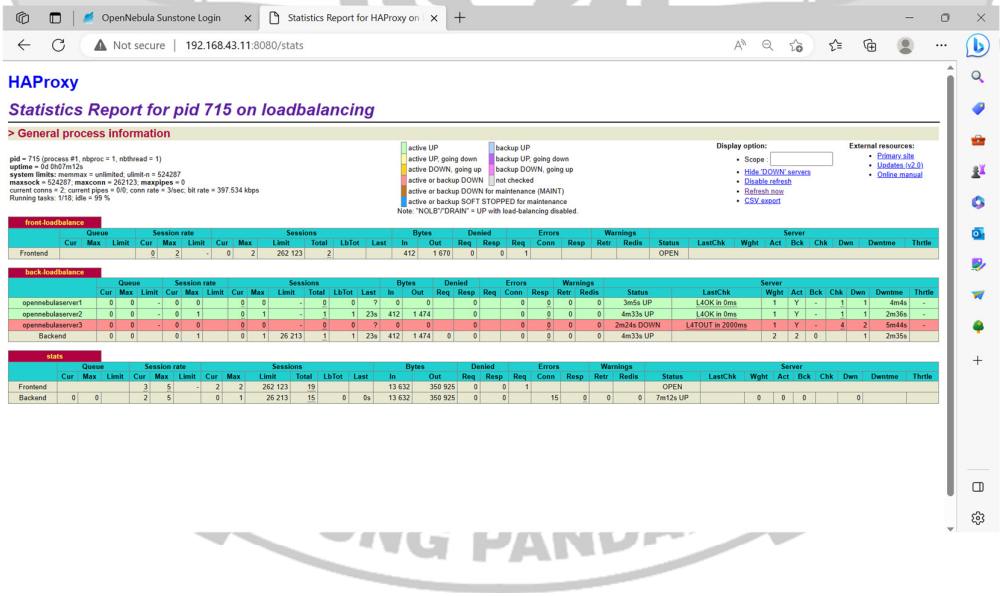
```
syahrir@opennebulaserver1:~$
syahrir@opennebulaserver1:~$ sudo su - oneadmin
[sudo] password for syahrir:
oneadmin@opennebulaserver1:~$
oneadmin@opennebulaserver1:~$
oneadmin@opennebulaserver1:~$ ssh-keyscan opennebulaserver1 databaseserver >> /var/lib/one/.ssh/known_hosts
# opennebulaserver1:22 SSH-2.0-OpenSSH_8.2p1 Ubuntu-4ubuntu0.5
# opennebulaserver1:22 SSH-2.0-OpenSSH_8.2p1 Ubuntu-4ubuntu0.5
# opennebulaserver1:22 SSH-2.0-OpenSSH_8.2p1 Ubuntu-4ubuntu0.5
# opennebulaserver1:22 SSH-2.0-OpenSSH_8.2p1 Ubuntu-4ubuntu0.5
# opennebulaserver1:22 SSH-2.0-OpenSSH_8.2p1 Ubuntu-4ubuntu0.5
# databaseserver:22 SSH-2.0-OpenSSH_8.2p1 Ubuntu-4ubuntu0.5
# databaseserver:22 SSH-2.0-OpenSSH_8.2p1 Ubuntu-4ubuntu0.5
# databaseserver:22 SSH-2.0-OpenSSH_8.2p1 Ubuntu-4ubuntu0.5
# databaseserver:22 SSH-2.0-OpenSSH_8.2p1 Ubuntu-4ubuntu0.5
# databaseserver:22 SSH-2.0-OpenSSH_8.2p1 Ubuntu-4ubuntu0.5
oneadmin@opennebulaserver1:~$ scp -rp /var/lib/one/.ssh databaseserver:/var/lib/one/
config                                                                                    100% 1444     1.0MB/s   00:00
known_hosts                                                                               100% 3822     3.0MB/s   00:00
id_rsa.pub                                                                                100%  580   509.5KB/s   00:00
authorized_keys                                                                           100%  580   417.8KB/s   00:00
id_rsa                                                                                    100% 2610     1.4MB/s   00:00
```

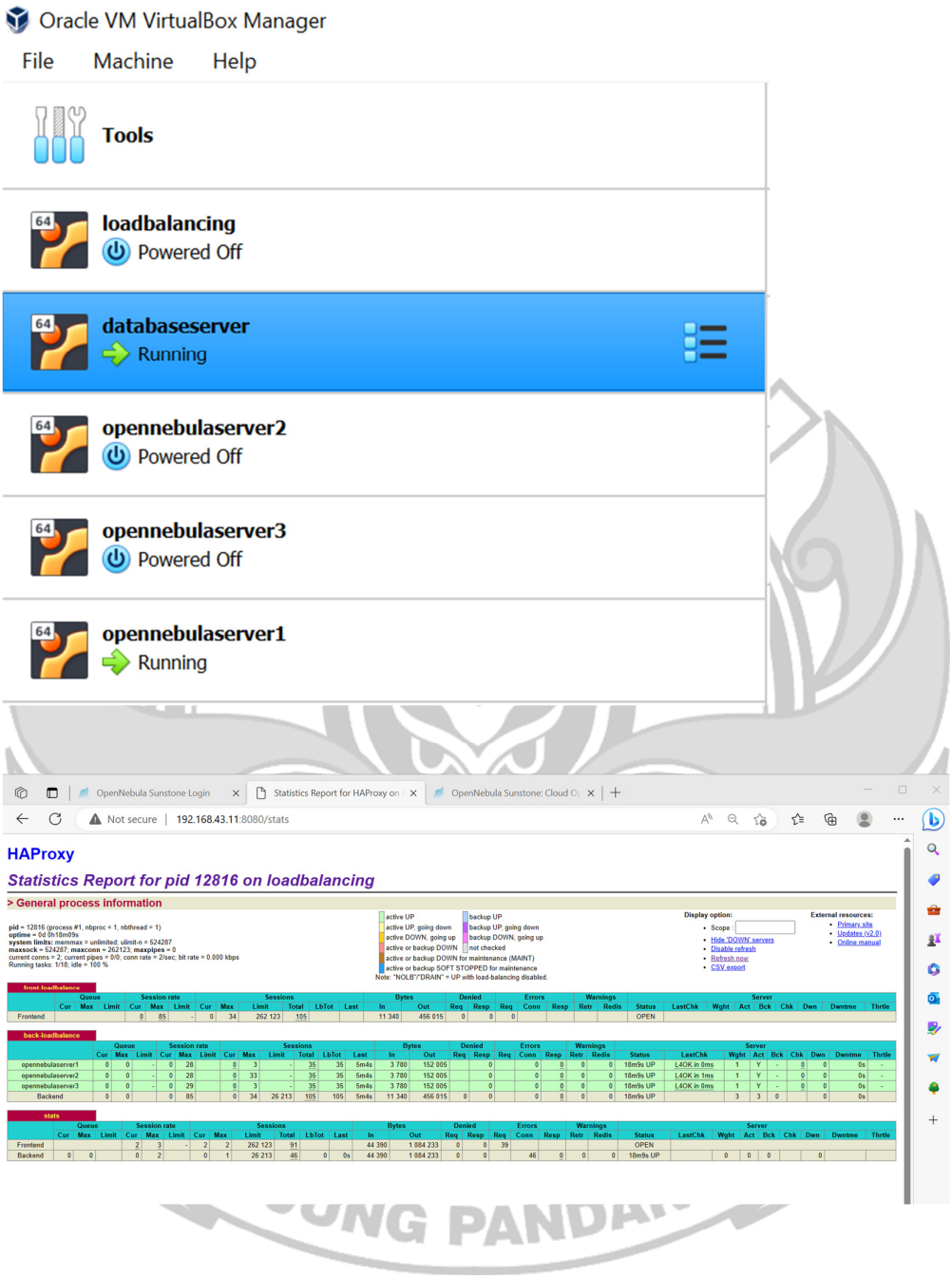


```
syahrir@opennebulaserver2:~$
syahrir@opennebulaserver2:~$ sudo su - oneadmin
[sudo] password for syahrir:
oneadmin@opennebulaserver2:~$
oneadmin@opennebulaserver2:~$ ssh-keyscan opennebulaserver2 databaseserver >> /var/lib/one/.ssh/known_hosts
# opennebulaserver2:22 SSH-2.0-OpenSSH_8.2p1 Ubuntu-4ubuntu0.5
# opennebulaserver2:22 SSH-2.0-OpenSSH_8.2p1 Ubuntu-4ubuntu0.5
# opennebulaserver2:22 SSH-2.0-OpenSSH_8.2p1 Ubuntu-4ubuntu0.5
# opennebulaserver2:22 SSH-2.0-OpenSSH_8.2p1 Ubuntu-4ubuntu0.5
# opennebulaserver2:22 SSH-2.0-OpenSSH_8.2p1 Ubuntu-4ubuntu0.5
# databaseserver:22 SSH-2.0-OpenSSH_8.2p1 Ubuntu-4ubuntu0.5
# databaseserver:22 SSH-2.0-OpenSSH_8.2p1 Ubuntu-4ubuntu0.5
# databaseserver:22 SSH-2.0-OpenSSH_8.2p1 Ubuntu-4ubuntu0.5
# databaseserver:22 SSH-2.0-OpenSSH_8.2p1 Ubuntu-4ubuntu0.5
# databaseserver:22 SSH-2.0-OpenSSH_8.2p1 Ubuntu-4ubuntu0.5
oneadmin@opennebulaserver2:~$
oneadmin@opennebulaserver2:~$ scp -rp /var/lib/one/.ssh databaseserver:/var/lib/one/
config                                                                                    100% 1444     1.6MB/s   00:00
known_hosts                                                                               100% 3822     4.2MB/s   00:00
id_rsa.pub                                                                                100%  580   913.2KB/s   00:00
authorized_keys                                                                           100%  580   198.1KB/s   00:00
id_rsa                                                                                    100% 2610     3.3MB/s   00:00
oneadmin@opennebulaserver2:~$
```

```
syahrir@opennebulaserver3:~$
syahrir@opennebulaserver3:~$ sudo su - oneadmin
[sudo] password for syahrir:
oneadmin@opennebulaserver3:~$
oneadmin@opennebulaserver3:~$ ssh-keyscan opennebulaserver3 databaseserver >> /var/lib/one/.ssh/known_hosts
# opennebulaserver3:22 SSH-2.0-OpenSSH_8.2p1 Ubuntu-4ubuntu0.5
# opennebulaserver3:22 SSH-2.0-OpenSSH_8.2p1 Ubuntu-4ubuntu0.5
# opennebulaserver3:22 SSH-2.0-OpenSSH_8.2p1 Ubuntu-4ubuntu0.5
# opennebulaserver3:22 SSH-2.0-OpenSSH_8.2p1 Ubuntu-4ubuntu0.5
# databaseserver:22 SSH-2.0-OpenSSH_8.2p1 Ubuntu-4ubuntu0.5
# databaseserver:22 SSH-2.0-OpenSSH_8.2p1 Ubuntu-4ubuntu0.5
# databaseserver:22 SSH-2.0-OpenSSH_8.2p1 Ubuntu-4ubuntu0.5
# databaseserver:22 SSH-2.0-OpenSSH_8.2p1 Ubuntu-4ubuntu0.5
# databaseserver:22 SSH-2.0-OpenSSH_8.2p1 Ubuntu-4ubuntu0.5
# opennebulaserver3:22 SSH-2.0-OpenSSH_8.2p1 Ubuntu-4ubuntu0.5
oneadmin@opennebulaserver3:~$
oneadmin@opennebulaserver3:~$ scp -rp /var/lib/one/.ssh databaseserver:/var/lib/one/
config                                                                                    100% 1444     1.5MB/s   00:00
known_hosts                                                                               100% 3822     3.5MB/s   00:00
id_rsa.pub                                                                                100%  580   797.1KB/s   00:00
authorized_keys                                                                           100%  580     2.6KB/s   00:00
id_rsa                                                                                    100% 2610   682.5KB/s   00:00
oneadmin@opennebulaserver3:~$
```

Lampiran 2 Dokumentasi Statistik Metode Failover Algoritma Berjalan




Oracle VM VirtualBox Manager
File
Machine
Help
Tools
loadbalancing
Powered Off
databaseserver
Running
opennebulaserver2
Powered Off
opennebulaserver3
Powered Off
opennebulaserver1
Running
OpenNebula Sunstone Login
Statistics Report for HAProxy on
OpenNebula Sunstone: Cloud O
Not secure
192.168.43.11:8080/stats
HAProxy
Statistics Report for pid 12816 on loadbalancing
> General process information
pid = 12816 (process #1, nbproc = 1, nbthread = 1)
uptime = 0d 0h18m09s
system limits: memmax = unlimited; ulimit-n = 524287
maxsock = 524287; maxconn = 262123; maxpipes = 0
current conns = 2; current pipes = 0/0; conn rate = 2/sec; bit rate = 0.000 kbps
Running tasks: 1/18; idle = 100 %
active UP
active UP, going down
active DOWN, going up
active or backup DOWN
active or backup DOWN for maintenance (MAINT)
active or backup SOFT STOPPED for maintenance
backup UP
backup UP, going down
backup DOWN, going up
not checked
Note: "NOLB"/"DRAIN" = UP with load-balancing disabled.
Display option:
Scope :
Hide 'DOWN' servers
Disable refresh
Refresh now
CSV export
External resources:
Primary site
Updates (v2.8)
Online manual
front loadbalance
back-loadbalance
stats